Automatic Generation of Human-Computer Interaction in Heterogeneous and Dynamic Environments Based on Conceptual Modelling

Von der Fakultät für Mathematik,
Naturwissenschaften und Informatik
der Brandenburgischen Technischen Universität Cottbus

zur Erlangung des akademischen Grades

Doktor der Ingenieurwissenschaften
(Dr.-Ing.)

genehmigte Dissertation

vorgelegt von

Diplom-Informatikerin
Jana Lewerenz
geboren am 11. Januar 1972 in Malchin

Gutachter: Prof. Dr. Bernhard Thalheim
Gutachter: Prof. Dr. Tiziana Catarci
Gutachter: Prof. Dr. Heinz Schweppe

Tag der mündlichen Prüfung: 14. Dezember 2000

Contents

List of Figures

List of Tables

Introduction

Interaction is one of the prevailing concepts in everyday behaviour. The dictionary calls interaction a 'mutual or reciprocal action or influence' [Col91]. This definition means that interaction *(a)* involves several interaction *partners*, *(b)* manifests itself in diverse *activities* and *(c)* that there is an *interplay* between these activities. This is true for interaction in general and for human-computer (HC) interaction in particular.

Let us imagine a visit in a travel agency: You have brought along your kid because you want to sort out a family trip to Hathersage. Travel agent Jodie is there to help you. Jodie and you *interact*. Jodie talks to you to find out about your wishes and preferences for transport, accommodation and entertainment. She asks you questions; you answer. Jodie will then do her own 'processing' of the received information: She will consult travel guides and catalogues, make a few phone calls, execute bookings. Eventually she will return her 'results' to you: a travel schedule, tickets, confirmations, vouchers. In turn, you also ask Jodie questions to find out what else one could do; she answers. Having learnt you are a fan of English social life, Jodie would suggest to visit the Scotsman's Pack because it is the best pub in town. She also turns to your kid and asks: 'And you, little lady, do you like swimming?' Your kid nods fervently. 'Then there is a very nice heated pool in Hathersage ...'

This is interaction.

> Interaction is a collection of activities of several interaction partners together with a temporal and causal ordering of these activities within a defined scope.

Interaction activities in every day life can be extremely varied. They can involve anything from door banging to playing a love song. Most of the time, however, they are of a communicative nature: uttering questions and answers, making phone calls or using the computer. Temporal and causal orderings mean that activities are sequenced and can induce and influence each other: You answered to Jodie's questions, and Jodie answered to yours. The scope of an interaction defines its limits. In the travel agency example, we could identify several interaction occurrences, namely between Jodie and you, between Jodie and the

computer booking system and between Jodie and your child. Regarding the interaction occurrences between Jodie and you on the one hand and between Jodie and the computer booking system on the other hand, we can put both into their own scope and except for an interface (your answers that Jodie uses as input to the computer) there is no link between them. ¿From another point of view, we could put you, Jodie and the computer booking system along with your interactions into one scope which means that you have access to the interaction between Jodie and the computer and could, e.g., correct her if you watch her giving some wrong input.

The necessity of interaction. Why do people interact? Of course, interaction can be the result of coincidental events: of the accidental pressing of a computer key, of Jodie talking about Hathersage to another customer and you overhearing her, etc. Usually, however, interaction is a deliberate act and is basically initiated to widen one's 'scope of power'.

Interaction enables a person to enact activities previously out of his/her reach, to exploit external sources of knowledge, etc. You interacted with Jodie because otherwise you would have no access to any booking facilities and because you do not know about pubs in Hathersage anyway. It is the wish to widen our scope of power that makes us communicate, coordinate our activities, give directions, act and encourage (or discourage) others to act.

In HC interaction in particular, the driving force of interaction can be a human user's wish to retrieve, store or modify information or to control the behaviour of the computer application system in question. Still rather uncommon—but certainly increasing—is the possibility that HC interaction is initiated by the computer system in order to exploit human experts' knowledge, to notify human users of events, etc.

Cooperativeness and it's a big big world. We have pointed out that interaction is usually performed deliberately with a certain intention in mind. Hence, what one would naturally be interested in is the *successfulness* of interaction. Successfulness of interaction means that your intention, be it to get information, to make somebody act or make somebody believe something, is realised. Successfulness, however, is not for free.

We can interact in various ways and in order to be successful we have to choose that way which is most appropriate for the interaction situation in question. Interaction can differ in the following aspects:

Flow: The order of activities can differ. You could have asked Jodie about the best pubs of England and then organised your tour around those recommendations instead of the other way around.

Content: We can interact about different things. Instead of recommending pubs, Jodie could have given additional information on hill-climbing near Hathersage.

Form: We employ different modes and styles of interaction. Jodie could have led a written conversation with you via e-mail instead of verbal communication (mode). She also used different expressions when talking to you and your kid (style).

How interaction differs, depends on various factors. The interaction partner is one of the most prominent criteria: He/she might be interested in different things, might prefer different interaction strategies, different terminologies, etc. Other factors are the communication medium available, e.g., face-to-face communication vs. telephone communication, the interaction history, etc.

An interactive behaviour that takes into account the characteristics and preferences of the interaction partner and the resources currently available and acts accordingly is called *cooperative*. Cooperativeness is essential for the successfulness of interaction.

This is where the 'bigness' of our environment comes in and makes everything even more difficult. An interaction environment is the collection of all factors that can influence our way of interaction: characteristics and preferences of interaction partners, available resources (speech, images, text; size and time constraints, etc.), certain stylistic features we insist on (verboseness, language, etc.) and various others.

Most interaction environments are heterogeneous and dynamic and hence pose special challenges to interaction. *Heterogeneous environments* are characterised by a large number of possible instantiations (interaction situations), e.g., we have to cope with very different types of interaction partners and may have to manage with a differing availability of resources. In the case of *dynamic environments* we cannot be sure about possible situations that might evolve in the future, i.e., it is not only that we have to manage a large number of situations (this might be done with by diligence), but moreover new situations might appear that we have never thought of before, e.g., we meet a person who does not fit into any of our known 'categories' or we suddenly have to interact via gestures only.

Interaction in environments which involve a big number of possible interaction situations that are moreover subject to frequent changes, is less predictable and hence it is incomparably more difficult to prepare for cooperative interaction. The support of HC interaction in heterogeneous and dynamic computer environments is the topic of this work.

HC interaction in heterogeneous and dynamic environments. ¿From the point of view of cooperativeness, interaction in computer applications used to

be predictable and neat. This does not imply perfection but what we mean here is that computer applications were developed for a relatively well-known audience: office automation software for the employee, teaching software for the student, computer games for the enthusiast.

Networked environments have now led to the increasing penetration of computer applications into daily life. More and more user groups are attracted, more and more technical novelties are offered, more and more software is introduced. We encounter the shift from specifically trained users and a purpose-oriented computer infrastructure towards a colourful mix of different user types expecting to access information systems through whatever device they know and like.

The necessity to provide cooperative interaction, however, remains. But due to the great variety of users, their equipment, interests and preferences, cooperativeness is incomparably more difficult to achieve. If even in traditional interactive computer applications the development of HC interaction consumes an average of 45% of the design time, 50% of the implementation time and 37% of the maintenance time in system development [MR92], how much money and time will have to be spent to cater to an audience that surfs through the Internet and is more diverse than in the most metropolitan of cities?

The challenge. HC interaction is part of almost any computer application system. Thus, the development of an application system usually involves the development of HC interaction, i.e., its design and implementation. HC interaction is to be modelled, thus describing the shapes that it is intended to and will be able to take in its eventual implementation.

The most prominent challenge for the support of HC interaction in heterogeneous and dynamic environments is the provision of *modelling methods* to enable a description of HC interaction in all its expected flexibility—flexibility with regard to different interaction situations as well as to possible changes in the future.

State of the art. It is a well accepted fact that high-quality interaction between a computer system and its human users is vital for the acceptance and the effectiveness of the computer system in question [FL00, Dow91]. Interaction has received an even higher importance by the emergence of the interactive computing paradigm where Wegner coined the term 'harnessing the environment' to describe its significance [WG98].

Recognising this importance has led to the introduction of several models and architectures for the development of HC interaction. Examples of well-known and prominent architectures are the *Seeheim model* [PH85], the Model View Controller paradigm (*MVC*) [KP88] or the Presentation Abstraction Control architecture (*PAC*) [Cou87]. All these models will be described in detail in Section 2.4.2.

These models implement a distinction between application functionality, physical presentations and communication between these two aspects. Thus, they have realised a first separation of HC interaction-related concerns and purely application-related concerns. They allow HC interaction designers to identify information that is of relevance in a specific application context, to make it available to or receive it from the HC interface component and to have information appropriately presented to the user either for provision or elicitation.

What these models fail to do, however, is to offer explicit support for HC interaction in heterogeneous and dynamic environments, i.e., to acknowledge the fact that different interaction strategies, contents and presentations are required for different customers and situations. Although MVC and PAC do allow the designer to provide different presentations for specific application objects, this can only be seen as a first step into the required direction.

In reality, the degree of flexibility provided often is severely restricted by the fact that the correspondences between application objects and presentations must be manually determined and hard-coded in the implementation. Thus, the emergence of previously unexpected interaction situations requires a re-design of the application system—a costly practice. Moreover, this approach only offers flexibility with regard to interaction form. The responsibility to customise flow and content of HC interaction is strictly excluded from the HC interaction component of application systems.

Meeting the challenges. It was user-centered research which first pointed out that HC interaction is impeded by the fact that users vary in a number of characteristics and are ever less willing to accept high efforts in learning how to use a specific interface [MHC+96, SKM93]. First approaches solved this problem by shifting it to the application functionality or even by calling for the 'universal' interface:

- Some research suggested that HC interaction could be constructed as simple and intuitively usable as required as to also allow users without a specific background to use it [MHC+96].
- Alternatively, the need for adaption should be shifted from the user towards the HC interaction component of the application itself [SKM93]. The system is then able to automatically produce or select appropriate HC interaction flows, contents and presentations.

The first approach of simplifying HC interfaces—thus creating a 'universal interface'—is rejected in this work. It risks offending expert users who want and need powerful control at their hands which would most probably not be of interest—and therefore not be made available—to inexperienced users. In

the light of different interface devices to be addressed, this rejection becomes even stronger as sophisticated functionality offered by advanced interface devices should certainly be exploited in order to support efficient HC interaction.

The more promising strategy, which is also pursued in this work, is therefore to turn information systems into *information services*. The term 'service' suggests that the computer system presents information in a way that is tailored to the particular recipients and that it is not the user who is required to learn, to adapt and to be appropriately equipped. In other words, an information system becomes an information service if it is able to customise its behaviour and appearance to the environment it currently operates in. Typical examples of applications that could profit from this approach are city and tourist information systems, home entertainment, etc. [FST98].

Automatic interface generation has repeatedly been suggested as desirable in view of rising demands for productivity, efficiency and flexibility in application design [SE96]—a demand that is even more pressing in heterogeneous and dynamic environments. A number of prototypes have realised an automatic generation of HC interaction; a methodical support, however, is still missing.

The provision of methods to structure the process and components of interaction development and to determine the amount of control and flexibility required by a certain application allows a planned and efficient approach to HC interaction design. A methodology for the development of HC interaction for heterogeneous and dynamic environments must provide means and methods

- to develop a model of HC interaction that can serve as a common basis for the generation of HC interaction in different situations—situations that are characterised by different user requirements, technical equipments and stylistic preferences—,
- to define the factors that characterise relevant interaction situations and that potentially influence the way HC interaction is performed and
- to specify the effects of these factors on HC interaction.

Contribution of this thesis. This thesis addresses the above issues. It develops a framework (the ACE framework) which establishes a method for the conceptual modelling and the automatic generation of HC interaction in heterogeneous and dynamic environments. This method includes the proposal of modelling concepts to abstractly describe HC interaction and the provision of an exemplary modelling language. It also incorporates an architecture for the automatic generation of customised HC interaction (customised to the current interaction situation) based on such an abstract model; a case study points out effects of interaction situations on the results of generation. A prototype that is currently being implemented is used to validate this method in principle.

Above all, this thesis provides a *methodical contribution* to the development of HC interaction. While concrete implementations are (and should be) interchangeable, this work presents concepts underlying methods of abstractly modelling and automatically customising HC interaction.

The ACE framework. To cope with HC interaction in heterogeneous and dynamic environments efficiently, we need a model that identifies phases as well as aspects involved in the development of HC interaction—a model that offers designers the required and desired amount of control and flexibility.

This work presents the *ACE framework*—a powerful frame concept for the conceptual modelling and the automatic generation of HC interaction. It structures the different tasks and aspects in HC interaction design and relates them to each other. It provides HC interaction designers with the conceptualisation that is required to ensure and control flexibility in interaction modelling, to define influences on interaction and to automatically map the model of interaction onto concrete presentations. This work also puts forth approaches to the practical application of the ACE framework.

The ACE framework basically identifies two dimensions of HC interaction:

Abstractness of HC interaction considers the *phases* involved in HC interaction development. Abstractness moves from

- an abstract *interaction space* that includes a complete description of all interaction features and their conditioning (without, however, considering concrete presentations) to
- an *evaluation* where those features are selected that are of relevance in a particular interaction situation and further on to
- the eventual *presentation* which decides on the physical instruments of interaction.

Scope of HC interaction covers the *aspects* of interaction that the individual interaction features belong to. Scope is divided into

- the *flow* of interaction, i.e., the sequence of individual interaction steps,
- the *content* of interaction, i.e., the information to be exchanged, and
- the *form* of interaction, i.e., the appearance of interaction.

Each of these aspects is to be explicitly considered during each of the phases above.

The ACE framework is based on the principles of abstraction, completeness and explicitness. *Abstraction* refers to the fact that a level of interaction description is introduced which is neither concerned with technical details of the

computer infrastructure to be used nor with any purely stylistic issues, and that starting from this point interaction will then be concretised. *Completeness* means that both the abstract properties of HC interaction and the influences from the environment that are relevant to determine this eventual concretisation are described. *Explicitness* is understood as the call for an open specification of all properties and environmental influences so that they can be used for an automatic concretisation (generation); in the most optimistic case, explicitness ensures full executability of the developed HC interaction model, otherwise it is used to support its prototyping.

This work provides the following contributions:

Conceptual modelling: We establish and motivate constructs for the conceptual modelling of HC interaction. In contrast to existing approaches, we explicitly consider *all* components of interaction on an *abstract* level.

This is especially true for the *form* of interaction: The constructs for the modelling of interaction form are not tied to specific interface systems. This is a new aspect in the modelling of HC interaction; it is required to actually achieve high customisability of interaction.

Automatic generation: We define phases for the concretisation (i.e., the automatic generation) of interaction based on an abstract and complete conceptual model. These phases structure and guide the process of interaction concretisation.

Customisation: We discuss possible environmental influences on the automatic generation and their impact on the eventual presentation of HC interaction. This discussion provides guidelines for a customisation of HC interaction in practice.

Structure of the Thesis

Part I – Fundamentals. The first part of this thesis provides a general introduction into the area of HC interaction and the problems interaction has to cope with. This part also discusses related work.

In **Chapter 1** we describe the role and the properties of HC interaction and introduce some required terminology. In particular, we explore the challenges that arise in heterogeneous and dynamic interaction environments and the way in which they are of importance to the modelling and generation of HC interaction.

In **Chapter 2** we give an overview of approaches to the understanding and modelling of interaction. Above all, we explore the potential of several existing approaches to cope with interaction in heterogeneous and dynamic interaction environments. The approaches discussed are of various natures: they originate from

linguistics, sociology and computer science and provide general models and concrete implementations, respectively. What is important is that these approaches regard interaction from different points of view so that they can serve as valuable sources of inspiration.

Part II – The framework. In the second part of this thesis, we develop a framework for the abstract modelling and the automatic generation of HC interaction—the ACE framework. We also illustrate its use.

Chapter 3 provides a general overview of the ACE framework. The framework is based on the requirements formulated in Chapter 1 and draws motivation from the concepts and approaches discussed in Chapter 2. Chapter 3 describes the dimensions the framework comprises and outlines how the framework is put to use.

Chapter 4 then describes the concepts used for the abstract conceptual modelling of the HC interaction component of an application.

An overview of the process of HC interaction development within the ACE framework is provided in **Chapter 5**. We pay special attention to the abstraction layer Co-Design model, i.e., we will embed HC interaction modelling into the different layers of the model and discuss its integration with other aspects of application modelling.

Chapter 6 presents an architecture for the automatic generation of HC interaction based on an abstract model. It also includes a case study that explores possible translations of HC interaction models into concrete presentations depending on the actual interaction situation.

Part III – An application. The third part of this thesis considers aspects of the practical implementation of the developed framework. It illustrates how the proposed framework has been partially implemented in the Co-Design prototype as one possible application.

Chapter 7 describes the Co-Design prototype. We outline its underlying ideas and explain its architecture and a number of important concepts it applies.

In **Chapter 8** we present the language and the methods used for interaction modelling.

Finally, in **Chapter 9** we explain in more detail different versions of the driver concept that the Co-Design prototype uses for the automatic generation of HC interaction.

Part I

Human-Computer Interaction: Challenges, Goals, Achievements

Over the years, more and more attention has been paid to the field of HC interaction. It has been recognised as one of the key factors in achieving usability and acceptance of computer systems by its human users. Undoubtedly, the majority of computer systems *is* designed for the use and the control by human users and should, therefore, give serious consideration to the provision of appropriate means of interaction to these users.

In the following, we explore the field of HC interaction, define the tasks it has to fulfill and point out the problems it has to cope with. Several perspectives and approaches to interaction are explored.

Chapter 1 provides an introduction into the area of interaction in general. First of all, we provide a basic definition of interaction and describe the tasks which the support of interaction has to fulfill and the instruments that are employed to solve these tasks.

We then introduce and explore the notion of interaction environments. Here, we point out which factors can influence the way interaction is performed and which effects these factors could have.

After this discussion we pin down the requirements of interaction—requirements both of a universal nature and requirements that emerge in the presence of heterogeneous and dynamic environments. A modified definition of interaction, taking into account the need for an adaption of HC interaction, is then provided.

Eventually, we concentrate on *human-computer* interaction and characterise the role of HC interaction in the context of information systems. We focus on the application system type of *high-level information systems* and will discuss the process of information system development in some detail, with particular attention given to the aspect of HC interaction.

Finally, we clearly state the scope as well as the delimitations of this work.

Chapter 2 then presents fundamental and related work. This includes some basic notions like different communication models and interactive computing as well as research on dialog systems and models and prototypes of graphical interface systems.

We discuss the contribution of these models and approaches to the development of HC interaction and critically analyse their suitability for coping with HC interaction in heterogeneous and dynamic interaction environments.

Chapter 1

Tasks, Requirements and Problems of Human-Computer Interaction

Since the departure of computer systems from mere number crunching, HC interaction has emerged as a discipline that has effect on almost all other uses of computer technology. It should be difficult to find many sensible applications that are not designed for the interaction with human users, be it for information retrieval, information gathering or some kind of control of the system behaviour.

For the acceptability and the usefulness of computer systems it is decisive that, given a certain functionality a system advertises, users can accomplish *what* they want and *how* they want it. Of course, this is to a large extent a matter of correct implementation of the promised functionality; but none the less important are the means of HC interaction that are expected to give efficient access to the system's functionality.

In order to fulfill these expectations it is necessary to clarify some fundamental notions of interaction. The following chapter puts forth the conceptual foundation for our further work. Based on a basic definition of interaction (Section 1.1), it provides an insight into the tasks and instruments of interaction (Section 1.2). We introduce the notion of interaction environments (Section 1.3) and then derive requirements for interaction (Section 1.4). After these discussions, we will develop a modified definition of interaction which takes the need for adaptability and customisation of interaction into account (Section 1.5). We look at the process of information system development and the role HC interaction development plays there (Section 1.6) and finally state the scope of this work (Section 1.7).

1.1 A First Definition of Interaction

What is interaction, what does it want and how does it work? These are the first questions we have to answer when we want to establish a framework for the conceptual modelling and automatic generation of HC interaction.

First of all, we will provide a basic definition of interaction to give the background required for a discussion of some fundamental notions of interaction. After the introduction of heterogeneous interaction environments, this first definition will be extended so that the aspect of adaptability and customisation of interaction can be covered as well.

Principally, we view interaction as a collection of activities that transform entities which are either real world objects (transforming an open door into a closed door) or information objects (transforming your vague ideas of a trip to Hathersage into a precise travel schedule). The interaction model describes these transformations along with the entities and the interaction partners involved.

Definition 1 *An* interaction model M *is a 5-tuple* (P, E, T, E_0, E_F) *with*

- P *being the set of interaction partners* p*,*
- E *being the set of interaction entities* e *(i.e., information or objects that are to be transformed by interaction) including a null element,*
- $T = \{t = (p_{Perf}, P_{Rec}, e_{Start}, e_{End})\} : p_{Perf} \in P,\ P_{Rec} \subseteq P,\ e_{Start} \in E,\ e_{End} \in E$*, being the set of transformations* t *described by a performing interaction partner* p_{Perf}*, a set* P_{Rec} *of receiving interaction partners, a start entity* e_{Start} *and an end entity* e_{End}*,*
- $E_0 \subseteq E$ *being the set of initial entities and*
- $E_F \subseteq E$ *being the set of final entities.*

There must exist a path from every initial entity to a final entity so that T *must satisfy the following condition:*
$\forall e \in E_0 : \exists\ T' \subseteq T$ *such that* $T' = \{t_i = (p_{Perf,i},\ P_{Rec,i},\ e_{Start,i},\ e_{End,i})\} :$
$e_{Start,1} = e \wedge e_{End,i} = e_{Start,i+1} \wedge e_{End,n} \in E_F;\ i = 1, .., n-1.$

By describing specific interaction partners and all possible activities between them, an interaction model prescribes the scope of interaction as mentioned in the introduction. Causal and temporal orderings are reflected in the concatenation of transformations.

Whereas an interaction model describes all possible occurrences of concrete interaction, an interaction instance defines one individual path within the interaction model that is either actually or potentially being taken.

Definition 2 *A* interaction instance I *within an interaction model* M *is defined as a 2-tuple* (M, T_I) *with*

- M *being the interaction model described by the 5-tuple* (P, E, T, E_0, E_F) *and*
- $T_I \subseteq T$ *with* $T_I = \{t_i = (p_{Perf,i},\ P_{Rec,i},\ e_{Start,i},\ e_{End,i})\} : e_{Start,1} \in E_0 \wedge e_{End,i} = e_{Start,i+1};\ i = 1, .., n-1$, *being the sequence of transformations actually or potentially performed.*

Interaction instances are inherently non-deterministic; which path will be taken cannot be known at the time of modelling. Moreover, interaction instances do not always lead to a final entity, i.e., are not always successful.

The term '(HC) interaction' that will be used in the remainder of this work, subsumes at least one (but often several) actual or potential (HC) interaction instances.

1.2 The Tasks and Instruments of Interaction

This section is intended to familiarise with the methods we can employ to realise interaction which means above all to perform the required transformations. We will describe the tasks as well as the processes involved in the initiation and the execution of the transformation of interaction entities.

1.2.1 The Tasks of Interaction

For illustration let us return to the travel agency example and the interaction executed there: ¿From Jodie's point of view the interaction starts with a null entity since when you enter the agency she does not know anything about your wishes. She knows, however, what sort of agency she works in and that you are likely to ask for a trip to be organised. So she will try to transform the null entity into an entity describing your requirements: place, time, number of people and any preferences you might have in terms of transport, accommodation, etc. This entity she will then attempt to transform into various bookings and tickets while feedback or follow-up questions are usually required.

Thus, the most interesting part are the transformations between entities. How are these realised? First of all, Jodie needs to offer you the possibility to *provide information to her*; she does this by asking. A computer system would offer you an entry form. Next, she will need to *provide information to you herself*, e.g., in the form of a printed travel schedule. These are the initial and the final transformations. In between, however, the steps of providing you with information and having you provide information are often *combined*. Jodie would, e.g., say that

there is a choice between a room with a view towards the Scotsman's Pack or towards the moorlands—and you would tell her what you prefer.

To realise such transformations, interaction has to fill two roles: the role of mediation and the role of transmission. Figure 1.1 shows an illustration of these roles.

Interaction in its mediating role is responsible for the transport of concepts, that have a particular meaning, between the interaction partners. This means the passing of information (such as Jodie's information about two choices and your information about your selection) from one partner to the other.

Interaction in its transmitting role is responsible for the physical transport of data representing such concepts. Unfortunately, humans are not yet able to directly communicate concepts but have to do so by the use of symbols which must be produced and eventually be interpreted to extract their meaning. So, Jodie has to utter a sentence containing her proposition and you have to utter a response.

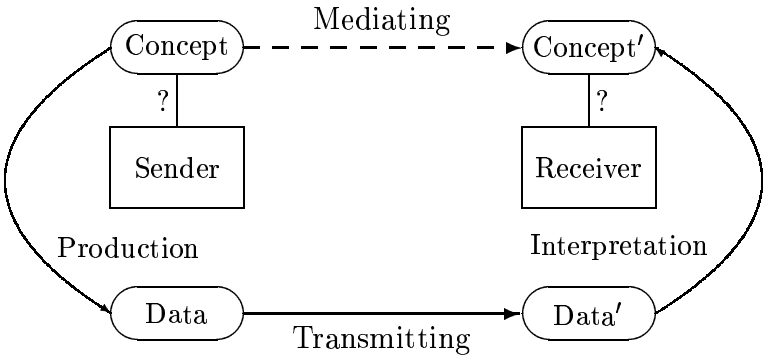


Figure 1.1: Transmission and Mediation

The impossibility of transporting concepts directly makes the use of the intermediate level of transmission necessary. In HC interaction, transmission is enabled by the provision of interfaces which are used to provide information to the user and extract information from the user. The gap between mediation and transmission requires translations: on the sender's side concepts must be translated into data suitable for transmission (production); on the receiver's side data must be translated back into concepts (interpretation). Obviously, both translations are not necessarily equivalent so that interaction becomes a somewhat less predictable process and requires careful and methodical planning and execution. To ensure that the receiver's interpretation corresponds to the sender's intention

(Concept = Concept'), the sender needs to take the receiver's way of interpretation into account, i.e., the sender must *customise* what he/she says and how it is said to the receiver. This fact is the incentive behind our approach to a conceptualisation of HC interaction.

What do we do in order to fulfill the two roles of mediation and transmission in an optimal way? In the following, we explore basic concepts that concern the organisation of interaction to better understand the processes involved.

1.2.2 The Components of Interaction

We can identify three components of interaction: We interact in a particular order, exchanging particular information (concepts) and using particular representations (for transmission). These are the components of flow, content and form.

Interaction flow is the overall structure of interaction. It considers the way interaction contributions are directed (i.e., their sender and their receiver) and the order in which they are exchanged.

Interaction content describes the information exchanged between interaction partners by means of interaction contributions. This description involves the type, i.e., the structure, of contributions as well as their values, i.e., their instantiation.

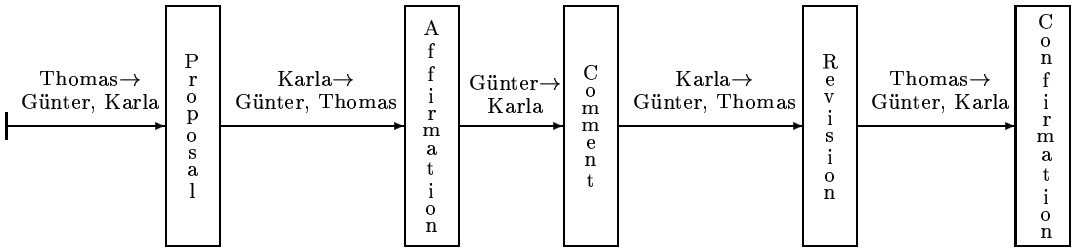
Interaction form is the appearance that is assigned to the information exchanged. It can refer to visual, auditive, tactual or other characteristics, depending on the mode of interaction.

Let us consider the following interaction instance as an example:

$$
\begin{aligned}
&I = (M, T_I)\\
&T_I = \{t_1,\ t_2,\ t_3,\ t_4,\ t_5\}\\
&t_1 = (Thomas, \{Günter, Karla\}, null, Proposal)\\
&t_2 = (Karla, \{Günter, Thomas\}, Proposal, Affirmation)\\
&t_3 = (Günter, \{Karla\}, Affirmation, Comment)\\
&t_4 = (Karla, \{Günter, Thomas\}, Comment, Revision)\\
&t_5 = (Thomas, \{Günter, Karla\}, Revision, Confirmation)
\end{aligned}
$$

This example is illustrated in Figure 1.2. The interaction instance shown above could correspond to the following situation: Thomas proposes to book rooms in a hotel for the three interaction partners; Karla sends her agreement to both Thomas and Günter but Günter then tells Karla that this hotel is situated in a very busy street; Karla then revises the proposal to another hotel; eventually, Thomas sends the final confirmation.

The respective flow for this interaction is shown in part (a) of Figure 1.2. At the source and the target of each arrow the interaction entities concerned can be found. An entity at the start of an arrow is provided to the performing actor and is transformed into the entity at the end of an arrow. Part (b) exemplarily describes the type of the entity Proposal; in part (c) we see an instantiation; and part (d) shows a possible presentation of the entity of type Proposal.



x→ y$_1$, ···, y$_n$: Interaction partner x performs a transformation of entities with the result being available to interaction partners y$_1$ to y$_n$.

Part a) Interaction flow

```
Proposal = Record (h: Hotel,
                   p: Place
                   r: Set (Record (n: Number,
                                   t: Type)))
```

Part b) Interaction content: A data type

```
p: Proposal
p = (Bologna, Pisa,
     {(1,single),(1,double)})
```

Part c) Interaction content: An instantiation

I propose to book one single room and one double room at the hotel Bologna in Pisa.

Part d) Interaction form

Figure 1.2: Examples of Interaction

1.2.3 The Process of Interaction

The components of interaction discussed in Section 1.2.2 represent a synchronic view of interaction, i.e., they can be used to describe the structure of interaction at a given point of time. E.g., the examples given in Figure 1.2 represent interaction at a very concrete stage where we have already decided on what to say in which order and in which form.

However, there is also a diachronic perspective on interaction. This diachronic perspective describes the processes involved in the translation between meaning (concepts) and symbols (data): production and interpretation.

For the production of interaction we can identify the phases of knowledge, selection and presentation where we describe what is *possible*, then what is *applicable* and eventually how it is *represented*.

Knowledge means that, e.g., humans acquire—either by experience or by instruction—knowledge on how to behave in a variety of situations. We have pointed out in the introduction that interaction is influenced by the current situation, i.e., on the specific goal to be achieved, on the interaction partner to be addressed, on the media available, etc. Consequently, humans usually possess knowledge on possible lines of persuasion (interaction flow), on possible types and instantiations of information exchanged (interaction content) and on possible stylistic properties like verboseness, terminology, etc. (interaction form).

Selection then means to select a subset of those rules that is taken as being relevant and applicable in a particular situation. This also involves an analysis of the current situation (partner to be addressed, media available, stylistic defaults, interaction state and history).

Presentation finally means the mapping of the relevant and applicable concepts onto a concrete physical form, i.e., to realise a particular sequence of contributions, to utter concrete words or draw pictures, to manipulate voice characteristics, etc.

The step from knowledge to selection is still a purely internal ('mental') effort. It consists of the evaluation of the interaction situation and the application of selection rules to the set of known concepts.

The step from selection to presentation, however, crosses the boundary between the mental model and some physical form. It is characterised by a number of semiosic phenomena, i.e., phenomena that concern the representation of (abstract) concepts by (concrete) *signs* [Eco97] (cf. Page 90).

1.3 Interaction Environments

Interaction is always set within a certain context. A number of factors will influence the way interaction is performed, i.e., in the case of HC interaction, the way it is designed and implemented [Mye93]. The entirety of these factors is what we call an *interaction environment*. A concrete instantiation of the factors of an interaction environment is called an *interaction situation*.

1.3.1 Factors in Interaction Environments

In the following we will describe various factors of interaction environments. Figure 1.3 shows some typical environmental factors.

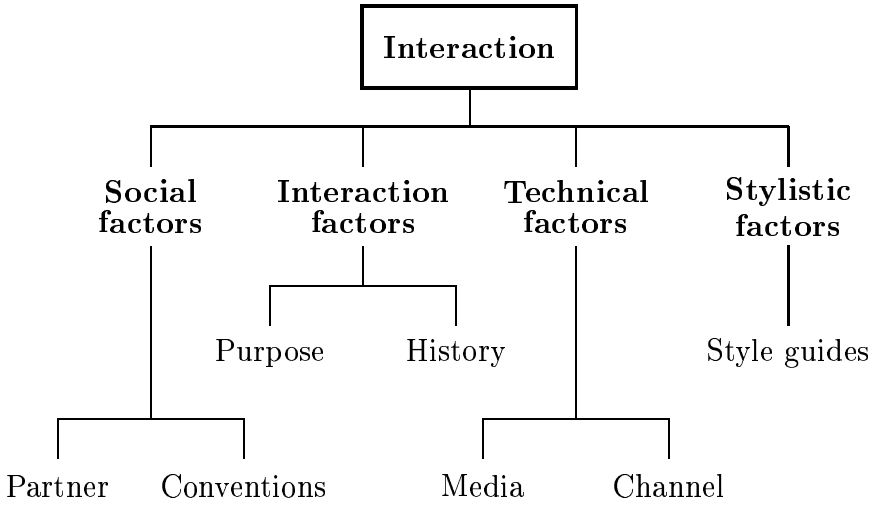


Figure 1.3: Factors of Interaction Environments

The social factors. Properties of the *interaction partners* involved as well as *conventions* that influence the structuring of interaction or the production and interpretation of interaction contributions form the social aspects of interaction environments.

Properties of *interaction partners* can include individuals' levels of expertise, personal preferences, cognitive abilities, interests, languages known or possible physical handicaps. Altus [Alt94] has provided a profound overview of characteristics that are of potential interest to interactive tasks.

Interaction *conventions* describe protocols that have to be followed in interaction. These protocols mostly originate from the specific cultural or domain-oriented background. An example of such protocols in human interaction is the structure official letters are expected to have (greeting – concern – elaboration – closing – enclosed documents). Conventions can also provide contextual information for the use of metaphors, colours or symbols as well as expectations in terms of politeness, e.g., conventional forms of greeting, set phrases, etc. A break of conventions can lead to amusement in the best case or to serious misunderstandings and offences in the worst case. A relatively well-known example is the fact that in Bulgaria nodding means 'No' and shaking the head means 'Yes'. Many gestures that are definitely harmless in some countries (such as pointing the palm upwards and bending the index finger to ask somebody to come closer) can have

a vulgar meaning in other regions. Several cultures associate shapes or colours with meaning that is rather unexpected for foreigners (such as triangular shapes having a negative aspect in Hong Kong and some other countries) [Ano94].

The technical factors. Another important influence on interaction are the physical properties of the interaction environment. These concern the *media available* and the *interaction channel*.

Typical *media* in human interaction are oral conversations, written conversations, gestures, mimics, etc. In HC interaction we might have to deal with graphical interfaces, natural language interfaces, command line interfaces, virtual reality, etc. Interaction media can form hierarchies as, e.g., face-to-face conversations and phone conversations as a subtype of oral conversations or Tcl/Tk and Visual Basic as a subtype of graphical interface languages. Moreover, several media can be used simultaneously (oral conversations plus gestures and mimics, graphical interfaces plus natural language output, etc.). In the case of HC interaction, there is the additional criterion of *device capabilities* as, e.g., display size, monochrome vs. colour display, etc.

The *interaction channel* can be characterised in terms of its capacity, i.e., the size of a telegram form in the case of human interaction or the bandwidth in the case of HC interaction. The robustness and reliability of an interaction channel also plays a role.

The stylistic factors. Stylistic factors determine the appearance of interaction. Examples are the verboseness of interaction contributions, the sentence structure in natural language interaction, voice characteristics in speech, colour schemes and the arrangement of widgets in graphical interfaces.

Stylistic factors are established by the sender of an interaction contribution. In the case of HC interaction, this means that they have their origin in guidelines given by the client (corporate identity, etc.) or in the experience and the preferences of an interface design expert.

The social factors of an interaction environment do also, however, reflect stylistic properties, namely stylistic *preferences, expectations and requirements* of the interaction partner (the user in HC interaction). Unfortunately, the sender's (client's, designer's) and the user's preferences can contradict each other. The sender's wish to impress by a verbose interaction style might, e.g., contrast with the receiver's preference for straightforward, minimal interaction. Prescribed colour schemes can fail to pay attention to red-green colour blind users or to meanings associated to these colours in specific cultures.

The interaction factors. Besides rather external factors of interaction environments like social, technical and stylistic aspects, interaction is also influenced

by the intended *goal* of interaction and by information on what has happened so far, i.e., the *interaction history*.

The interaction *goal* determines the overall structure of information, the target that is aimed for. On this basis, possible flows of interaction can be developed that serve as a starting point for an adaption to the remaining external factors.

The interaction *history* contains an overview of what has been done so far. Above all it allows to judge on the successfulness of the adopted strategy and on the need to modify it. Moreover it can keep record of information already exchanged so that unwanted redundancy can be avoided.

1.3.2 Effects of Environment Factors

Each of the different factors of interaction environments has effects on the three components of interaction: flow, content and form. As an illustration we will now give an overview of possible effects.

Social factors:

- *Flow:* Users might prefer (due to their personal properties) or expect (due to their cultural background or their experiences) certain strategies of interaction or certain sequences of interaction steps like first a summary, then detailed information or vice versa.
- *Content:* Users' interests can make the use of different interaction entities appropriate, e.g., suggestions for hill climbing facilities instead of pub tours. The concrete instantiation of these entities can also depend on user characteristics, e.g., only cheap offers might be given to students.
- *Form:* Interaction form is again influenced by personal preferences or requirements of users (e.g., bright colours for the young and trendy, special colour schemes for red-green colour blind users) and by expectations due to their cultural background or their experiences (red colour for warnings, etc.).

Technical factors:

- *Flow:* The properties of the interaction channel can influence the amount of interaction, e.g., a slow channel makes the reduction of the number of individual interaction steps advisable.
- *Content:* In a specific interaction medium only certain types of entities might be representable so that a choice between, e.g., text and images can be necessary.

- *Form:* Media also have certain capabilities for the representation of interaction form, e.g., the capability to show colours, to influence the arrangement of information, etc.

Stylistic factors:

- *Flow:* Strategies for sequencing interaction steps can be given, e.g., to guide users to certain offers.
- *Content:* Similarly, the types as well as the instantiation of interaction entities can be controlled, e.g., to only show expensive offers of hill climbing facilities instead of cheap pubs.
- *Form:* The aspect that will be most influenced by stylistic factors is the interaction form, though. This includes the colour schemes to be used, the terminology, the language, the arrangement, the voice characteristics, etc.

Interaction factors:

- *Flow:* In HC interaction, the initiative for deciding on the flow of interaction can be with the system, the user or both (hybrid initiative). In the case of system-controlled initiative, the system can employ planning mechanisms to realise an interaction flow according to the user's goal. Planning mechanisms have especially been developed in the field of artificial intelligence [Pol92, BBD+93, YM94, Pol96].
- *Content:* The interaction history can give information on data that has already been exchanged and that can be assumed as being known to the user. Thus, unwanted redundancy can be avoided.
- *Form:* If redundant information is exchanged, e.g., as context information, its role must be made apparent by an appropriate form, e.g., a less prominent appearance.

Further influences of environmental factors are of course possible. The examples given are to serve as an illustration of possible effects.

1.3.3 Heterogeneous and Dynamic Environments

A trend in information systems is the shift from specifically trained users and a purpose-oriented computer infrastructure (e.g., in office automation) towards the wish or even requirement to offer public access to inexperienced, occasional users expecting to utilise information systems through whatever device they have available and/or prefer. Such a setting often results in a heterogeneous and dynamic environment and poses special challenges to the design of information systems in general and to the design of HC interaction in particular.

Heterogeneous environments are characterised by a number of different interaction situations to be addressed. This means the existence of several types of users (interaction partners) and associated conventions as well as types of interface devices (interaction media) and channels. Moreover, information often may have to be presented in different appearances (interaction styles).

Dynamic environments are characterised by the need to often modify or extend the interaction environment by new user types, interface device types and/or styles to be addressed. As a consequence, the way interaction is performed changes constantly while the actual message of interaction stays the same. Future changes of the interaction environment can often not be foreseen at the time of the initial development of an information system.

1.4 Demands on Interaction

After having introduced the basic notions of interaction that are relevant for this work, we will now explore which demands the support of interaction has to meet. In the centre of our attention is the term *cooperativeness*. Cooperativeness is essential in order to achieve the intended goal of interaction. It means that a style of interaction in flow, content and form is adopted that is tailored to the encountered interaction situation and that motivates and enables the interaction partner to correctly interpret as well as produce interaction contributions.

Providing a collection of generally valid principles, the language philosopher Grice has formulated the following *conversational maxims* [Gri75]:

Maxim of quantity:

Make your contributions to the conversation (interaction) as informative as necessary.

Do not make your contributions to the conversation more informative than necessary.

Maxim of quality:

Do not say what you believe to be false.

Do not say for which you lack adequate evidence.

Maxim of relevance:

Be relevant (i.e., say things related to the current topic of the conversation).

Maxim of manner:

Avoid obscurity of expression.
Avoid ambiguity.
Be brief (avoid unnecessary wordiness).
Be orderly.

To become more specific, we can derive the following requirements for interaction flow, content and form that are true for interaction in general:

- Interaction flow:
  - a logical interaction structure that is easy to follow
  - no mental jumps
- Interaction content:
  - an appropriate complexity of individual interaction entities that are used for information exchange
  - information fitting the actual interests and requirements of users
  - the inclusion of context information to allow for an orientation within interaction
  - survey and help information
- Interaction form:
  - clarity and as little ambiguity of presentations chosen as possible (a careful use of metaphors and other indirect symbols)
  - consistency of presentations
  - aesthetical quality of presentations

Heterogeneous and dynamic environments. Realising these requirements is often difficult enough but is made much more difficult in heterogeneous and dynamic environments. What is considered comprehensible and intuitive is in fact subject to the cognitive abilities, preferences and the background of individual interaction partners. The conventions that associate abstract concepts and concrete symbols (or rather signs) are anything but universal and even such a straightforward word like 'tree' might give rise to fairly different notions in regions like, e.g., Greenland and Brazil. Only by a careful and considerate use of interaction flows, contents and forms can misunderstandings be minimised and the efficiency of interaction be increased.

In the context of heterogeneous and dynamic environments we have, therefore, to cope with additional challenges:

- a *customisability* of interaction to cater to different interaction situations and
- an *extensibility* of interaction to support the incorporation of previously unexpected interaction situations.

Customisability. On the one hand, the mere existence of *several* user types and media types to be addressed and of several styles to be used is a serious challenge to the productivity of interaction development. It has, e.g., been shown that many Web users—who are usually considered a very innovative audience—are surprisingly conservative both in terms of software used and in terms of accepting new interaction techniques and styles [Nie98]. Even relatively primitive textual Web browsers like Lynx are still so popular that even today some Web sites reengineer their Web pages for higher browsing comfort in Lynx, e.g., dict.leo.org in April 2000!

For a public information system to be successful it is, therefore, indispensable to cater to a wide range of equipments and user expectations—e.g., conservative and innovative—and to provide different presentations for equal contents. The automatic generation of interaction can be used to create different dialog structures and appearances, ideally on the basis of a common interaction model. Research into this area has yielded some interesting results already (see Section 2.4.3 for an overview) but existing tools and design methods do net yet show the required degree of spontaneous flexibility. Moreover, adaption is mostly restricted to the form of interaction and does rarely involve interaction flow and content. Style guides for the incorporation of general principles of software ergonomics, HC interface design and others (Corporate Identity, etc.) are not automatically included in the development of HC interaction.

Extensibility. On the other hand, it is not feasible anymore to restrict HC interaction design to a known and predictable environment. Independently from the specific application domain, automatic interaction generation must be easily extensible to new user types, interface device types and styles. Especially interfaces types are an area where every now and then new paradigms (from early command line interfaces to current direct manipulation interfaces to—perhaps—virtual reality in the future) are developed. This leads to the need for the common interaction model to abstract as much as possible from any particular known and existing interaction paradigms.

Relevance of customisation. Methods that allow the creation of different interaction flows, contents and forms make a customisation of interaction possible. It has been argued that customised interaction is a decisive criterion in business

contacts (business to business as well as business to customer) [FL00]. Customised interaction allows for relationship marketing that targets customers individually by providing tailored offers (contents), lines of persuasion or modes of navigation (flow) and appearances (form).

Relationship marketing is vital in retaining 'old' customers and attracting new customers. It is an additional means of distinction besides—possibly—price and value-added services. Especially in view of the fact that the current value of many Internet businesses is based on investors' belief of their future cash-generating potential, satisfaction of customers must be considered one of the top targets.

Of special importance is the possibility of a *run-time* customisation of interaction because it is often not possible to describe or even to foresee all possible interaction situations. Run-time customisation is achieved through an automatic generation of HC interaction that takes the actual interaction situation into account. It allows for a consideration of current and detailed aspects of the situation actually encountered.

1.5 A Modified Definition of Interaction

In Section 1.1 we introduced a first definition of interaction which was required as a background for the discussion of the basic notions of interaction. Now, after having introduced the concept of heterogeneous and dynamic interaction environments, we will provide a modified definition of interaction which is able to accommodate the flexibility required in such environments.

Definition 3 *An* interaction environment *Env is a collection of attributes and their values. Attributes refer to possible social, technical, stylistic and interaction factors.*

Attribute values change over time and can be observed from the outside.

With regard to an interaction environment, requirements (conditions) can be established which the interaction environment is expected to fulfill at a certain point of time.

Definition 4 Requirements *Requ on an interaction environment Env are a logical connection of equations, inequations, set-equations and other types of equations over values of environmental attributes, values of interaction entities and constants.*

Definition 5 *A* satisfiability function $Sat : Requ \times Env \times Time \rightarrow \{0, 1\}$ *is a function which produces the value* 1 *if at a given point of time* $b \in Time$ *the requirements Requ are fulfilled in the interaction environment Env. Otherwise it produces the value* 0.

The syntax of requirements on interaction environments as well as the satisfiability function, which defines the semantics of requirements, can be based on common models such as predicate logics, etc. [SA91]. It is not the intention of this work to prescribe a certain model.

Definition 6 *Given an interaction environment Env, an* interaction model M *is a 5-tuple* $(P,\ E,\ T,\ E_0,\ E_F)$ *with*

- P *being the set of interaction partners* p*,*
- E *being the set of interaction entities* e *including a null element,*
- $T = \{t = (p_{Perf},\ Start,\ End,\ Requ_{Perf})\}$ *being the set of transformations* t *with*
  - $p_{Perf} \in P$ *being the performing interaction partner,*
  - $Start = \{(E_{Start},\ Requ_{Start})\}$ *being the set of pairs of a set* $E_{Start} \subseteq E$ *of start entities and requirements* $Requ_{Start}$ *on the interaction environment whose fulfillment will select* E_{Start}*,*
  - $End = \{(E_{End},\ P_{Rec},\ Requ_{End})\}$ *being the set of 3-tuples of a set* $E_{End} \subseteq E$ *of end entities, a set* $P_{Rec} \subseteq P$ *of receiving interaction partners and requirements* $Requ_{End}$ *on the interaction environment whose fulfillment will select* E_{End} *and* P_{Rec} *and*
  - $Requ_{Perf}$ *being requirements on the interaction environment whose fulfillment will decide on the executability of the transformation,*
- $E_0 = \{\{e : e \in E\}\}$ *being the set of initial entity sets and*
- $E_F = \{\{e : e \in E\}\}$ *being the set of final entity sets.*

What is new in contrast to the first definition of interaction is the use of *requirements* to determine the executability of transformations as well as the interaction entities they consume or produce and the receiving interaction partners. By means of these requirements the interaction environment can be taken into account. The executability of a transformation can depend on the existence of the required start entities as well as on various other properties of the interaction environment. If a transformation is found executable, start entities and end entities are also selected according to the current interaction situation. Another extension is the introduction of *sets* for start entities and end entities. This allows for a more complex branching of transformations (cf. Section 4.3) Finally, the condition on the *existence of paths* from initial entities to final entities has been dropped. This is due to the fact that the execution of transformations now depends on the interaction environment which is observable but not predictable

so that at the time of modelling, statements concerning the actual executability of transformations cannot be made.

The modelling language proposed in Section 4 will reflect this modified definition of interaction.

Definition 7 *Given an interaction model M, an interaction environment Env and a satisfiability function Sat, an* interaction instance *I is a 4-tuple $(M,\ Env,\ Sat,\ T_I)$ with*

- *M being the interaction model described by the 5-tuple $(P,\ E,\ T,\ E_0,\ E_F)$,*
- *Env being the interaction environment,*
- *Sat being the satisfiability function and*
- $T_I = \{t_i = (p_{Perf,i},\ E_{Start,i},\ E_{End,i},\ b_i)\} :$
  $p_{Perf} \in P \wedge E_{Start,i} \subseteq E \wedge$
  $E_{End,i} = \{(E_{End,j},\ P_{Rec,j}) : E_{End,j} \subseteq E \wedge P_{Rec,j} \subseteq P\} \wedge b_i \in Time,$
  being the sequence of transformations actually or potentially being performed, having consumed and produced certain interaction entities. T_I must satisfy the following condition:

$$
\begin{aligned}
&T_I = \{t_i = (p_{Perf,i},\ E_{Start,i},\ E_{End,i},\ b_i)\} : \\
&\quad (\exists t = (p_{Perf},\ Start,\ End,\ Requ_{Perf}) \in T : \\
&\qquad Sat(Requ_{Perf},\ Env,\ b_i) = 1 \wedge \\
&\qquad \exists (E_S,\ Requ_{Start}) \in Start : \\
&\qquad\quad (E_S = E_{Start,i} \wedge Sat(Requ_{Start},\ Env,\ b_i) = 1) \wedge \\
&\qquad \exists X \subseteq End : \forall (E_{End},\ P_{Rec}) \in E_{End,i} : \\
&\qquad\quad \exists (E_E,\ P_R,\ Requ_{End}) \in X : \\
&\qquad\qquad (E_E = E_{End} \wedge P_R = P_{Rec} \wedge Sat(Requ_{End},\ Env,\ b_i) = 1)) \wedge \\
&\quad E_{Start,1} \in E_0 \wedge \\
&\quad i = 1, ..n)
\end{aligned}
$$

Obviously, the condition on sequencing transformations according to their produced and consumed entities has been dropped in this modified definition. This is due to the fact that the sequencing of transformations is now controlled by requirements on the interaction environment ($Requ_{Perf}$).

1.6 HC Interaction Development for Information Systems

The focus of this work is the development of HC interaction. The type of application systems we are interested in above all are so-called *high-level information systems*. These are characterised by

- extensive use of database technology,
- a high amount of interaction to be realised and
- a medium amount of functionality to support the storage, retrieval, manipulation and processing of information.

Typical examples of information systems are office automation systems, city information systems for tourists, booking and reservation systems, etc.

1.6.1 The Development of Information Systems

There are four interrelated aspects involved in the design and implementation of an information system: the database schema definition, the view definitions, the behaviour and the interaction [CLT97]. All four aspects are shown in Figure 1.4 where they are related to the dimensions of locality and dynamics.

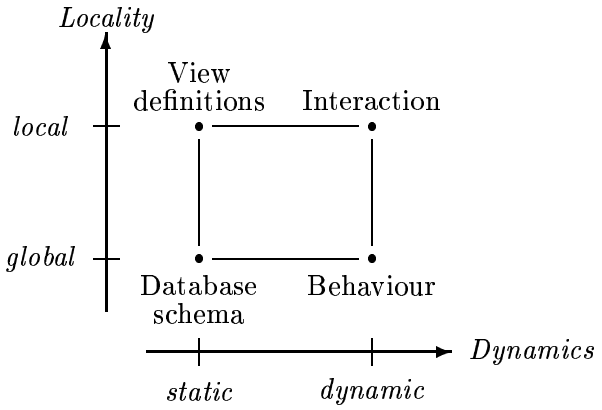


Figure 1.4: The Aspects of Information System Development

The database schema is the global, static aspect of system development. It comprises type definitions and the collection of all static integrity constraints. By means of the database schema possible database instantiations are determined.

Database views locally[1] present specific static instantiation parts to the user. The definition of views over views is possible so that view hierarchies can be created.

[1]Intuitively, the local aspect of information systems can be equalled to an *individual 'perspective'*, i.e., the perspective an individual user in a specific context has on the system.

The behaviour includes all possible functionality within the information system, in particular all dynamic integrity constraints (which by the way may also reflect the static integrity constraints). The behaviour thus captures the global and dynamic aspects of system design. Those functions currently being performed represent an instantiation of the behaviour definition.

Interaction corresponds to a local presentation of behaviour in a specific context to an individual user. Through interaction users are supplied with functionality and data (view instances). It involves the selection as well as the structuring of behaviour and a description of the data required for interaction. Complex interaction flows can be defined over primitive interaction contributions or other complex interaction flows.

The understanding of each of the above aspects is rather heterogeneously developed. Whereas the static parts (database schema and view definitions) are well understood and provided with a sufficient theoretical background [AA93, AHV95], the dynamic part is still lacking a unifying design methodology. A profound insight into and a methodical approach to the design of local behaviour (interaction) has almost completely been neglected.

Between the individual aspects of system development, mutual dependences exist as indicated in Figure 1.4. Disregarding these dependences can lead to inefficient, incomplete or incorrect designs [CT97]. Especially the design of HC interaction has, however, mostly been proposed as a separated activity [CW93, Ebe94]. Such an approach is often subject to difficulties in terms of efficiently mapping the required functionality into interactive behaviour on the one hand and flexibly catering to user and medium requirements on the other hand. The integration of interactive behaviour into information system development has only recently been addressed [BFJ96, CCK+94, Sch96].

1.6.2 The Development of HC Interaction

This work focuses on the aspect of HC interaction. HC interaction allows human users the access to an information system, i.e., to the underlying database and the functionality available. Typical activities that are to be supported are

- the search or querying for information,
- the survey of information,
- the storage of information,
- the browsing through an information space,
- the navigation between information,

- the processing of information, etc.

HC interaction is often expected to offer access to an information system for users without expert knowledge. This is achieved by masking the underlying data structures and/or offering a task-oriented guidance. Over and above, HC interaction should require a minimal learning effort.

Mapping application requirements into HC interaction. For the development of HC interaction, methodologies and technologies have to be provided that enable a mapping of constructs of the (real or envisioned) application domain onto control structures and constructs of a particular target interface system [CLS99b]. Potential target interface systems include a wide range of graphical systems (e.g., XView, HTML for graphical browsers), text-oriented systems (e.g., HTML for Lynx browser), interfaces based on forms, natural language (e.g., telephone interfaces, etc.) or systems of any other kind.

Both the requirements of an application and the HC interaction to be developed are expressed by means of (more or less formalised) languages. Thus, the mapping of constructs can principally be understood as a translation between these languages. The mapping or translation of the application requirements into control structures and constructs of the targeted interface system is intended to present and hence make accessible to the human user *(a)* information that plays a role in the application either for a provision to the user or for an extraction from the user as well as *(b)* corresponding activities for the storage, retrieval, manipulation or processing of that information. The mapping of HC interaction involves the three components of HC interaction: flow, content and form. With regard to all of these aspects, abstract requirements of the application are to be translated into concrete constructs at the HC interface end.

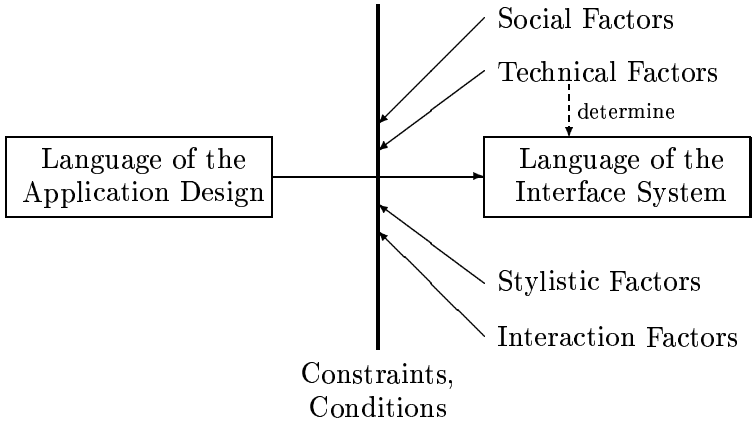


Figure 1.5: HC Interaction Development as a Mapping of Languages

Figure 1.5 illustrates the process of language mapping from the language of the application design (model) into the language of the targeted interface system. Both languages are characterised by a specific vocabulary and grammar, i.e., the constructs available and the rules for combining them. The language of the application design will contain constructs for data description and process description; the language of, e.g., graphical interface systems includes a specific number of widgets.

The translation can only result in constructs that are available in the language of the interface system. In addition, the eventual translation is subject to constraints and conditions that are posed by social, technical, stylistic and interaction factors of the interaction environment.

- Social factors constrain the translation to constructs the user understands or prefers (native or known languages) and the visual/auditive properties he/she can perceive or prefers (colour-blindness, etc.); cultural conventions determine the use of colours and other metaphors; cognitive abilities require an adaptation of the amount of information given at one time, etc.

- Technical factors (besides the actual interface system in question that is described by the language of the interface) can restrict the amount (due to low bandwidth) or the type of information (e.g., due to the inability to display graphics) that can be exchanged.

- Stylistic factors define constructs, colours, arrangements, etc., that are to be preferred out of stylistic reasons in order to create a particular impression.

- Interaction factors control above all the selection of accompanying information used as context information.

Integration of HC interaction with other development aspects. Developing the HC interaction aspect of an application system is occupied with the identification of data and functionality that are relevant for interaction and with their mapping onto physical representations. It is with regard to the identification part that a link to the system development aspects of database views and behaviour (cf. Section 1.6.1) becomes apparent.

On the one hand, information to be exchanged with the user often corresponds to database views. On the other hand, functionality required for the processing of that information naturally has to be a part of the overall system functionality, i.e., the system behaviour. During the design phase, the three aspects of interaction design, view design and behaviour design can complement each other to identify relevant objects, thus ensuring consistency and completeness of the entire design.

Modelling the environment. The design of HC interaction also incorporates the modelling of the interaction environment, i.e., of the social, technical, stylistic and interaction factors that may influence the organisation and the appearance of HC interaction. As a consequence, the behaviour as well as the appearance of an application system can be customised in order to particularly suit the environment encountered and to improve the usability of the application system. For the HC component of a system this means to answer the question

> *'When, where and to whom should which information be presented in what manner?'*—

'when, where and to whom' describing the current interaction environment,

'which information' describing the selected type and instances (contents) of information and

'what manner' describing the flow and the physical appearance of HC interaction.

The incorporation of these factors into the process of the development of HC interaction can be either static or dynamic.

In the case of *static* incorporation of environmental factors, HC interaction will be designed with respect to a particular fixed set of characteristics of the environment, e.g., a specific audience or a specific interface system. The resulting HC interaction will not be able to show any degree of adaptivity to changes in the environment. Often in such cases, the modelling of the environment will be implicitly conducted during the design phase; the specific characteristics of the targeted environment will not be made explicit in the HC interaction model.

Adaptivity of HC interaction can be achieved if properties of the environment are *dynamically* incorporated. In such a case, the domains of the individual environment factors must be specified and rules for the handling of specific instances of environmental factors must be given.

A dynamic incorporation of environmental factors usually involves an (at least partial) *automatic generation of HC interaction*. Automatic interaction generation is to heighten the productivity of system development by reducing implementation time and costs devoted to the interaction component of a system. By means of automatic interaction generation several possible instances of interaction environments can be addressed based on one common model of HC interaction.

1.6.3 Perspectives on HC Interaction

Interaction comprises a number of transformations of interaction entities. In the following, these transformations are also called *interaction processes* and interaction entities are often simply called *data* or *information*. We consider an

interaction process as consisting of the consumption of information by one interaction partner, the production of a response and the intermediate (mental) processing required to transform the consumed information into the response. Speaking in computer science terms, an interaction partner receives input, 'processes' the input and finally generates some output as response. Interaction is usually composed of several such interaction processes.

Naturally, the respective interaction partners have different perceptions of how interaction processes are formed. What is sent as *input* by interaction partner Agnes to interaction partner Ben, is perceived by Ben as *output* from Agnes.

Agnes's point of view		Ben's point of view
Input from Agnes to Ben	=	Output from Agnes to Ben
		⇓ Processing
Output from Ben to Agnes	=	Input from Ben to Agnes
⇓		
...		

As a consequence, the interaction partners' perspectives of interaction processes interleave. Figure 1.6 illustrates this interleaving of perspectives from the information system's and the user's point of view.

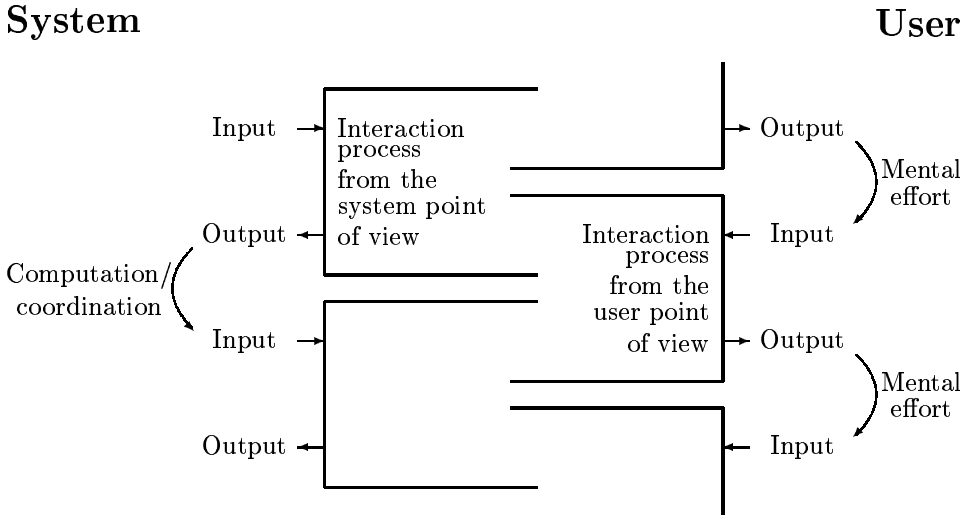


Figure 1.6: Perspectives on Interaction

For the modelling of HC interaction, we will take the information system's point of view when realising HC interaction, i.e., we will view an interaction process as providing input information to the user and in return receiving output

information from the user after some mental processing on the user's side. It has to be noted that the terms 'input' and 'output' thus are used in reverse to common usage. This perspective is, however, useful for the integration of interaction processes with other, namely computational, processes that the information system is to perform. We can now adopt a view of interaction processes that is analogous to that of computational processes which allows for a smooth integration of the two.

The difference between interaction processes and computational processes is that, while in the latter case the processing is executed on the basis of some known algorithms, in the case of interaction processes the respective algorithms are not (and generally cannot be) known to the information system. Instead it has to directly depend on the human user's mental capacity to (from the system's view point 'magically') perform the transformation of information entities.

1.6.4 Demands on Human-Computer Interaction

It is HC interaction that makes an interaction between a computer application system and its human users at all possible. To give optimal support to the information exchange between system and user, HC interaction is subject to the same requirements that have been identified for interaction in general (cf. Section 1.4). Speaking in general terms it has to

- be easy to understand, to follow and to use from the user's point of view (which is not necessarily the developer's),
- be self-explanatory and intuitively usable, i.e., to require minimal learning effort,
- offer clear access to the desired information,
- map the promised functionality in an obvious way,
- meet common and/or personal expectations in terms of interaction mechanisms and appearance,
- tailor its behaviour and appearance to the interaction situation encountered.

Human users are bound to using HC interaction as it is offered by the application system. Even with the ability to adapt to different interaction situations, HC interaction is, however, still far from achieving the degree of flexibility and at the same time robustness that can be expected in human interaction. Both from the users' and the developers' points of view, this is a blessing and a curse at the same time.

For the user, a restriction of possible activities and acceptable inputs to the system helps to prevent misunderstandings and can be a guidance as to how the system is to be used. On the other hand, the HC interaction offered might not fit the user's expectations, abilities and preferences and can thus be extremely difficult to use.

For the developer, the opportunity to express type constraints for user input and to exert control on the functionality offered decreases the possibility of incorrect use of the application system by the user. However, high requirements in terms of fitness for use have to be fulfilled. The developer needs to adopt different users' points of view in order to be able to cater for as diverse an audience as possible. Another aspect is that recent research gives proofs of computers often being accepted as social actors by their human users [NST94]. The expectations in terms of a sensible and robust intercourse are correspondingly high. Especially novice users often overrate the reliability of computer applications and are discouraged by errors that result from 'non-standard' (standard by the developers' conceptions) use of the interface [LN99].

1.7 Scope and Delimitation of this Work

This work is set in the context of HC interaction for high-level information systems. It provides a methodical *framework* for the conceptual modelling and the automatic generation of HC interaction for information systems in heterogeneous and dynamic environments. The emphasis is not so much on an actual realisation of the mechanisms proposed but rather on the development of those mechanisms itself. The main reason for such an approach is the wish to keep the mechanisms flexible enough to be incorporated into as many existing methods for the development of information systems as possible. Nonetheless, we have implemented a crucial part of our ideas and conceptualisations in a prototype which will be presented in Part III of this thesis.

The modelling of HC interaction. We will propose a set of concepts for the abstract modelling of HC interaction—the term 'abstract' meaning that at the point of modelling a specific interaction environment to be addressed is not fixed upon. These concepts have to capture the flow, the content and the presentational semantics (form) of interaction. They are intended to be suitable for the incorporation into a range of existing methods for information system development.

For presentational purposes, an exemplary modelling syntax will be provided.

The modelling of the interaction environment. With regard to the modelling of the different aspects of an interaction environment, we will discuss

which properties of the environment are of relevance and how they may possibly be extracted.

For the actual generation of HC interaction the existence of environment models is assumed.

Automatic HC interaction generation. The methods used for the automatic generation of HC interaction depend on the specific methodologies used for HC interaction modelling and environment modelling.

In this work, we will describe a rule-based mechanism for the automatic generation of customised HC interaction. We will furthermore provide a case study of possible physical presentations of HC interaction and of the influences that different properties of the interaction environment can have upon these presentations. In the context of the implemented prototype, we will present one possible implementation.

Chapter 2

Fundamentals and Related Work

The field of human-computer interaction has for a long time attracted much attention. It has yielded valuable conceptual as well as practical insights into the aspects involved in the development of HC interaction. Guidelines for the design of HC interaction have been developed and a number of prototypes have shown the feasibility and practicability of dialog systems, automatic HC interaction generation and other issues.

The problem of heterogeneous interaction environments has firstly been addressed with regard to different user types to be catered to. Much research has been done in the field of user modelling since the late seventies. Other aspects of heterogeneous environments like different interaction paradigms to be used (graphical vs. text-based interfaces) have started to attract interest only recently which is also due to the increased pervasiveness of networked environments.

Moreover, there exist several fields (mostly from linguistics) that offer fundamental contributions to the understanding of what is going on in interaction. These can provide welcome inspiration for the development of general interaction frameworks.

2.1 Communication Models

Activities of a communicative nature are ubiquitous in interaction. Several communication models capture the aspects involved in communication and relate them to each other. The communication models presented in the following each have different objectives and thus regard communication from different points of view.

The Lasswell Formula

The Lasswell Formula [Las48] defines the following scheme for the analysis and/or performing of communication:

'Who says what in which channel to whom with what effect?'—

'who' describing the sender of a message,

'what' describing content (meaning) and form of a message,

'which channel' describing the medium used for conveying a message,

'whom' describing the receiver(s) of a message,

'what effect' describing the results achieved by sending a message.

Being a sociologist and working in the fields of political propaganda and mass communication, Lasswell was particularly interested in the effect that could be achieved by communication and in ensuring efficiency of communication. His formula does not, however, provide any explicit means to evaluate the achieved effect as it does not capture the effect actually intended.

On the other hand, Lasswell is well aware of the importance of the interaction environment. The success of communication, i.e., achieving a particular effect, does depend on several factors: the sender may have properties (idealism, independence, etc.) that make him/her particularly credible or popular; one channel may be less credible than another (e.g., voice transmission only vs. simultaneous voice and image transmission); even the message itself can either repulse or attract depending on the style chosen. And behind all these factors one finds the receiver of the message: It is the receiver's characteristics that decide just what is acceptable to him/her and what is not, what is credible, what is appealing, etc.

We could also express the Lasswell Formula by means of a hierarchy of execution levels as is shown in Figure 2.1. The lowest level of influences on the execution corresponds to the interaction environment; it encompasses the *'who'*, the *'which channel'* and the *'whom'* aspects. The middle level of physical execution includes the *'what'* aspect, i.e., the actual content and form of the message. The highest level of results of the execution captures the *'what effect'* aspect, i.e., the feedback received from the receiver of the message.

One drawback of Lasswell's Formula is that it does not explicitly distinguish between the abstract content (meaning) that is to be communicated and the concrete form that is used for a message. Both aspects are mixed in the notion of *'what'*. As a consequence, the process of choosing a particular message form in dependence on the actual interaction situation is not an explicit aspect of the Lasswell Formula. Interaction structure, i.e., the relationships between individual messages, receives only minimal consideration in the Lasswell Formula. It is only the effect achieved that hints at some mutual exchange of messages.

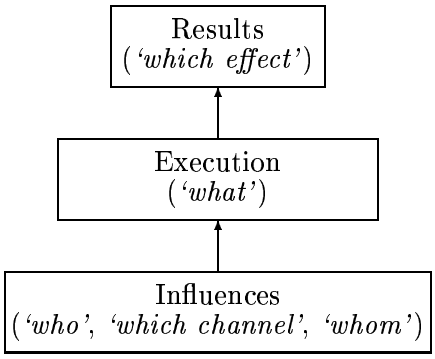


Figure 2.1: The Lasswell Formula as a Hierarchy

Shannon's Communication Model

The communication model that was developed by Shannon, an information theorist, mainly focuses on the process of message transmission [Sha48]. Figure 2.2 shows a graphical representation of the model.

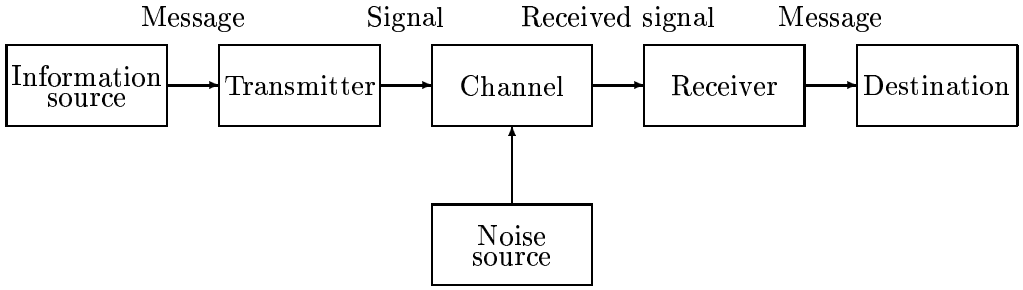


Figure 2.2: Shannon's Communication Model

The model includes the following components: An *information source* generates a message. A *transmitter* then transforms (encodes) this message into a signal that is suitable for transmission; the form of the signal depends on the channel chosen. The signal is transported over the *channel* which can potentially be disturbed by a *noise source* that could change the encoded signal. After transmission the signal is decoded into a message by the *receiver*; the decoding is usually an operation that is inverse to the encoding. Eventually, the decoded message is transferred to the intended *destination*.

Shannon's intention was to provide a model for accurate message transmission, i.e., for a reduction of the effects of the noise source. He achieves this by introducing redundancy into the signal to be transmitted. This redundancy helps in 'repairing' possible errors in the received signal and to reconstruct the original

message. Shannon's model is technical throughout. It is not concerned at all with the meaning of messages. Instead he uses the notion of *entropy* that describes the degree of unpredictability of an information source. Information is then a means to reduce this uncertainty.

For our purposes, Shannon's communication model is of rather philosophical interest only. It provides no means to express the general structure of interaction, i.e., the relationships between individual messages. The distinction between message content and form is absolutely technical and does not consider semantics. Firstly, however, the model can be used to emphasise that the form of interaction depends on the medium chosen for transmission. Secondly, when decoding a message, the capabilities to decode, i.e., the user's methods of interpretation, must necessarily be considered.

Bühler's Organon Model

Bühler, a linguist, developed the Organon Model [Büh82] that views language as a 'tool' used by *somebody* to communicate *to somebody else about the things—einer - dem anderen - über die Dinge* (cf. Figure 2.3).

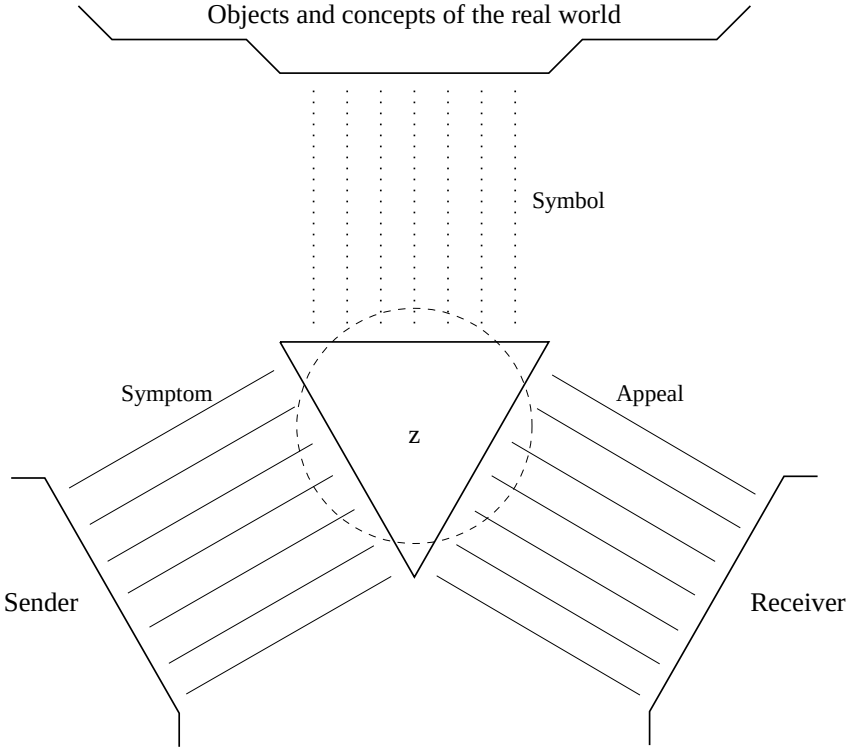


Figure 2.3: Bühler's Organon Model

In the Organon Model, the concrete sound phenomenon (Bühler concentrates on spoken language.) functions as a sign z in three different ways, thus manifesting the three different roles of language: The sign is a *symbol* since it refers to objects and concepts of the real world; it is a *symptom* since it expresses the sender's inner conception (his/her *Innerlichkeit*); it is an *appeal* since it controls the inward and outward behaviour of the receiver.

The Organon Model captures well the relation between content (meaning) and form of a message: A sign is chosen to represent objects and concepts of the real world. Whereas the objects and concepts to be conveyed are the content of a message, a sign constitutes its form. The influence of the environment is likewise acknowledged: As an appeal, the sign must be appropriate to the receiver's characteristics; as a symbol it must take existing conventions for presentations into account; as a symptom it is chosen according to the sender's stylistic preferences.

As Bühler concentrates on spoken language, the influence of different media is not covered. Similarly to the Lasswell Formula and Shannon's communication model, the Organon Model offers no means for capturing the flow of interaction.

2.2 Interactive Computing

The interactive computing paradigm has experienced a theoretical treatment by Wegner in cooperation with Goldin [Weg97, WG98, WG99]. It introduces the concept of *interaction machines* and is seen as an alternative and improvement to algorithmic computing that incorporates the concept of Turing machines as one of its central notions.

Interactive computing is not concerned with the physical realisation of interaction, i.e., the generation of interfaces, etc. Rather, its main goal is the establishment of a sound theoretical basis for the modelling of interactive activities. In that context, interaction is not confined to HC interaction but includes interaction with all possible external systems—computer applications, humans, sensors, etc. Interactive computing underlines the unpredictability of interaction and the forms it employs and thus is an important conceptual motivation for this work.

Computing by interacting. Turing machines serve as a model of computation by algorithmic programming. They transform predetermined input strings into output strings. During the process of transformation they shut out the external world and are not able to consider activities or even results of activities that had not been foreseen or had not been known at the start of that transformation. Clearly, in a real-world environment this is a serious drawback.

Interaction machines, however, act as transducers of interactively generated streams. They can make use not only of inner but also of external resources to solve problems. As a consequence, although they might possess less internal thinking power, they are able to replace self-sufficient inner cleverness by exploiting their environment. In an extreme case they might even have no intelligence of their own but behave intelligently 'only' by replicating intelligent inputs from their environment. And yet, interaction machines scale up to very large problems better than Turing machines because they can use the mechanisms of delegation and coordination. Wegner calls this mechanism *harnessing the environment* and compares it to the advantages of an open-book examination over a closed-book examination—from the performer's point of view.

Wegner defines two types of interaction machines: single-stream interaction machines (SIMs) and multi-stream interaction machines (MIMs). Whereas Turing machines are only able to produce algorithmically computable outputs, single-stream interaction machines can take former interactions with one interaction partner into account while computing, and multi-stream interaction machines can coordinate multiple contexts and support the collaboration of actors.

Expressiveness and observers. The behaviour of an interaction machine can be observed by several external observers (users, testers, external applications, etc.). Although the interaction machine can coordinate the streams of several observers, the observers themselves might be unaware of each other and of their activities so that the output from the interaction machine can appear subjectively non-deterministic to them. This is due to hidden variables in the case of SIMs and due to hidden interfaces in the case of MIMs. Any output is, however, still objectively deterministic from the point of view of an omniscient multi-agent observer (which Wegner equals to God).

The expressiveness of interaction machines is determined by the observing capabilities of the respective observers. E.g., a Turing machine observer can only relate single question-answer pairs while a SIM observer can observe the sequential behaviour of a single interaction stream.

Interaction that can be specified is not necessarily observable. E.g., the complete behaviour of multi-stream interaction machines is specifiable but hardly observable.

Mathematical models. The core concept for the mathematical foundations of interactive computing is *co-induction* which is—as opposed to induction—a technique for the definition and reasoning over non-enumerable circular structures like graphs and streams.

Induction is characterised by an *initiality condition* (describing the starting point of induction), an *iteration condition* (describing the derivation of new el-

ements from existing elements) and a *minimality condition* (ruling out any elements that cannot be derived in this manner).

Co-induction, on the other hand, includes an *iteration condition* (ensuring circularity) and a *maximality condition* (stating that all elements are possible that are not explicitly ruled out). Initiality is eliminated so that dynamic environments can be incrementally supplied.

The mathematical foundations of interactive computing are not yet completely developed. Once they are, they can serve as a theoretical model for human-computer interaction, too.

2.3 Dialog Systems

Dialog systems have been defined as

> computer systems with which humans interact on a turn-by-turn basis and in which *spoken natural language* plays an important part in the communication [Fra97][1].

Dialog systems are commonly equalled to communication systems involving two partners (the system and a human users) although the problem of more than two interaction partners is also addressed [HS94]. In the latter case, systems have to manage different belief spaces, i.e., models of their users' intentions and goals. Systems with two partners can be compared to single stream interaction machines (SIMs), whereas systems with more than two partners can be considered as multiple stream interaction machines (MIMs, cf. Section 2.2).

Spoken dialog is valued as a natural mode of communication for humans and as a motivation for more imaginative human-computer interaction [McT98]. It is of great importance for application systems wishing to offer access to disabled users, for so called eyes busy/hands busy applications, for telephone-based communication, etc.

Today's dialog systems surpass simple question-answer sequences and attempt to model coordination and relations between dialog contributions. Spoken dialog systems usually involve the following components [WSA+96]:

speech recognition where an input speech utterance is converted into a string of words,

language understanding where the string of words is analysed (interpreted) with the aim of producing a representation of the meaning of the utterance,

[1] Note that the term *dialog system* is also sometimes used to refer to graphical user interfaces.

dialog management where the cooperation between the different components as well as the interaction between the system and the user are controlled,

communication with external system(s) (e.g., a database, an expert system or any other external application system),

response generation where an internal specification of the message to be delivered is produced (text, concatenation of pre-recorded speech elements, etc.),

speech output where the internal specification is converted into speech output.

Figure 2.4 shows such a typical dialog system architecture. This dialog system architecture also bears resemblance to the Arch model which is discussed in Section 2.4.2.

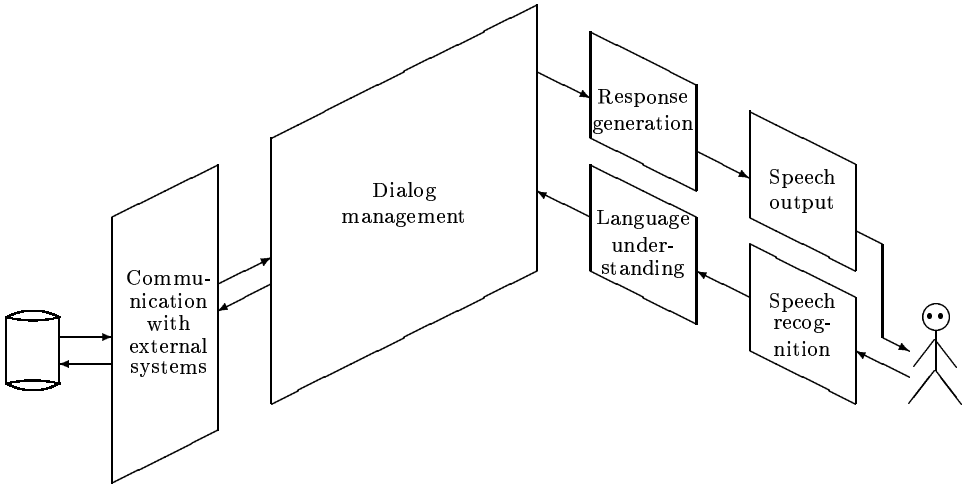


Figure 2.4: Dialog System Architecture

Of particular interest to this work are the aspects of dialog management and response generation. Dialog management models the structure of interaction using common techniques like state-transition nets, grammars or task-based and plan-based approaches, respectively. The response generation often uses vast background knowledge on the application domain, the user and the dialog history in order to produce tailored responses.

The **Corinna** system, e.g., is concerned with the generation of cooperative system responses in information retrieval dialogs [FMS94]. It is based on the COR model that provides a domain-independent dialog model of information seeking dialogs by means of a recursive state-transition diagram. This model contains conversation acts that occur in information seeking dialogs and describes possible

transitions between them. The COR model is enhanced by Rhetorical Structure Theory (RST) relations [MT87] (cf. Section 4.4.1) that provide an additional level of coherence description between dialog contributions. The combination of the COR model and RST relations is used to describe possible dialogs as well as the current dialog. Further semantic knowledge on the application area (information seeking) and the actual domain is then used to construct dialogs, i.e., to provide dialog contributions to the user.

Another very well-known dialog system is **HAM-ANS** [HCM+83]. Its domain are hotel reservation dialogs. Using heuristics it is able to resolve incorrect grammar, ellipses, uncertainties, etc. It can also infer new knowledge and summarise existing knowledge.

The **Verbmobil** project is concerned with the real-time translation of spoken natural language [Wah93] (cf. Section 4.4.1). Its main focus is, therefore, speech recognition and language understanding. What is interesting about the Verbmobil project is that in order to efficiently interpret utterances it tries to identify different dialog phases and dialog acts [ABF+97]. In the Verbmobil project, these serve as a context for the disambiguation of speech; they could also, however, be exploited by a dialog management component and response generation component to model the flow and the form of interaction. Dialog phases describe the general structure of interaction in certain interactive settings, e.g., hello - opening - negotiation - closing - good_bye for a negotiation dialog[2]. Dialog acts are associated to particular dialog phases and capture the intention of the utterances of dialog partners, e.g., suggest, confirm, give_reason, etc.

2.4 Graphical Human-Computer Interaction

In many application systems, HC interaction is of a graphical nature. Graphical HC interaction is appreciated as an intuitive means of interaction which is easy to learn and to use. In the following, we present a discussion of how the components of HC interaction (flow, content and form) are commonly modelled in graphical HC interface systems. We then describe a number of prominent system architectures as well as prototypes that implement the modelling and/or the automatic generation of HC interaction.

[2]Dialog phases are a good example of protocols that are determined by the user's cultural background as argued in Section 1.3.

2.4.1 Modelling the Components of HC Interaction

HC interaction involves the following components:

- the *flow* of interaction, i.e., the definition of individual interactive tasks and their sequencing,
- the *content* of interaction, i.e., the definition of interaction entities (data) to be exchanged between the application system and the user as well as their instantiation with concrete values and
- the *form* of interaction, i.e., the physical appearance of HC interaction contributions.

For each of these HC interaction components, different methods of modelling are available.

Interaction flow. Green describes three popular approaches to the modelling of interaction flow (dialogues in his terminology): transition networks, grammars and events [Gre86].

In a *transition network* several states of an interactive system are connected by actions that denote user activities and lead from one state to another. Several extensions like recursive transition networks and augmented transition networks exist.

By means of *(context-free) grammars* the language which is indirectly used by the user to communicate with the application system is described. Indirectly means that this language still has to be translated into physical representations of system output and into mechanisms for accepting user input.

The *event* model is based on the concept of input events. These events can be connected to procedures, e.g., computations, generation of new events, etc., that are invoked on the occurrence of the appropriate event. Events are, e.g., used in the MVC paradigm (cf. page 52) and the PAC model (cf. page 53).

Green argues that the event model has the greatest expressive power of the three and that the other two models can be converted into the event model. His claim with regard to the expressive power of the three models is mainly based on the fact that the event model can be integrated with (Turing-complete) programming languages so that the event model can be considered Turing-complete while the other two models are not.

Augmented transition networks, which include functions that perform computations on registers, have, however, been shown to be also Turing-complete so that this claim cannot be considered valid. Indeed, transition networks—along with events—have always been a very popular (if not the most popular) means for modelling the flow of interaction. This is mainly due to their intuitiveness

and their ability to capture the interaction flow at a more global level than events can do.

Interaction content. Common methods to model the content of interaction are the direct use of data models, the use of intermediate presentations or the specification of individual data objects.

Especially in the case of database applications or database-backed applications [JWZ93], *data models* like the entity-relationship model or the relational model are employed. This means that the schema and sometimes view definitions of the underlying database are used as the description of the interaction content. Other approaches map data models onto intermediate models such as graph models [CT95] and use these intermediate models to represent the data involved in interaction.

Based on simple database view definitions, the enhanced concept of *information units* has been developed for a user-adequate presentation and delivery of data in information systems [FST98]. In a first step, abstraction and rebuilding rules are used to abstract and derive new information (such as summaries) from a view. In a second step, a number of scaling rules are applied to customise this information and to provide further functionality. Scaling rules can sequence information, convert between units of measurement, specify adhesions between information or define hierarchies.

Another possibility is the specification of *data objects* [FKS95, MKB95]. Data objects are used to model data entities of the (real or envisioned) application domain—they work as 'containers' for concrete interaction data. A more sophisticated object-oriented model for describing the content of HC interaction has been developed by Schewe and Schewe [Sch96, SS00]. They propose the use of *dialog objects* to efficiently support the modelling of information which is relevant for interaction. Dialog objects consist of an abstract identifier, a set of values v_i in associated fields $F_1, \ldots F_n$, a set of references to other dialog objects (to allow for navigation), a set of actions to change data and to control the overall dialog, and a dialog state with possible values 'active' and 'inactive'. Dialog objects are described by dialog classes and are linked to the underlying database by means of view definitions.

The decision which of these options is more appropriate, depends on the framework used for the development of the application system. Whereas an object-oriented framework calls for the data/dialog object modelling method, other frameworks can well handle content specification based on ER models, the relationship model or other graph models.

Attempts to formalise the content of a user interface exist [RR98]. These are useful in the evaluation of customised interaction content, i.e., to ensure that no required information is lost during the customisation of HC interaction.

Interaction form. Mostly, the form of interaction is directly described by means of presentation objects or indirectly by means of the structure and/or semantics of content descriptions.

Presentation objects are associated with individual data objects and specify their physical appearance. Both the MVC paradigm (cf. page 52) and the PAC architecture (cf. page 53) use presentation objects. A number of prototypes also employ this technique [GMP+98]. By connecting one data object with several presentation objects, different physical appearances can be achieved. Presentation objects can be parameterised in order to visualise different instantiations of data or function objects [PTHM90].

Structure and semantics of interaction content can also be used to decide on the form of interaction. In this case, rules map data types used and/or data properties (attributes) [BFJ96, GWD90]. Data types are mostly mapped onto specific constructs, e.g., list types onto scroll lists, pull-down menus, etc.; attributes and semantics are mapped onto properties of constructs used for presentations, e.g., colour, position, size, etc. [DW91].

The second approach—using rules to map structure and/or semantics of interaction contents onto presentations—appears more appropriate, especially in the context of dynamic interaction environments. As the translation is not based on individual data objects in a specific context but on more general characteristics, it allows for the incorporation of new interaction paradigms without a complete re-engineering of the application system.

Combining the components of interaction and interaction environments. Next to the components of interaction (flow, content and form), the different factors of interaction environments also have to be modelled and incorporated into the process of HC interaction generation, i.e., the physical presentation of HC interaction components.

A very popular approach to the modelling and generation of HC interaction are model-based tools. An overview of such tools is presented by Schlungbaum and Elwert [SE96]. Model-based tools capture different aspects of the interaction part of a system in a declarative way in various models. These models interact with each other to generate HC interaction that suits the current interaction situation. Typical models are

- the application model (for the general behaviour of the application system),
- the task model (for the interactive tasks users have to perform),
- the user model (for the characteristics of users),
- the platform model (for the characteristics of the platform used, e.g., input and output devices),

- the workplace model (for the cultural context of interaction and other environmental characteristics),
- the dialog model (for the state, history and schedule of the interaction that has so far taken place and will take place),
- the presentation model (for the appearance of data and operations) and
- the behaviour model (for the input behaviour).

The integration of these interaction aspects (although not all of them are considered in all approaches) usually happens by means of an interaction controller that analyses the tasks to be performed and the current interaction environment, then decides on appropriate interaction flows, contents and forms and generates the corresponding physical presentations.

2.4.2 Architectures for Interactive Systems

Architectures for interactive systems develop a view of the main components that are required to implement interactive systems and of the relations between these components. They do not necessarily imply technical details; in fact, real systems or prototypes that adhere to any particular architecture may only logically but not physically implement the individual components.

In the following, a number of the most prominent and interesting architectures for interactive systems are described. In their evaluation, we pay particular attention to their suitability for heterogeneous and dynamic interaction environments, i.e., the possibilities they offer to keep the flow, content and form of interaction flexible and extensible. We examine how each of the architectures is able *(a)* to address different interaction situations at system run time and *(b)* to support modifications of the interaction environment that were not foreseen at the time of system design.

The Seeheim Model

The Seeheim model was proposed on a workshop in 1985 [PH85]. It describes the constituents logically required to realise the run-time part of a user interface management system (UIMS). Figure 2.5 shows a graphical representation.

- The *presentation component* is responsible for the immediate interaction with the user. It describes the HC interface on a *lexical* level by specifying the interface objects available. By means of these interface objects, the presentation component provides output to the user and accepts input from the user.

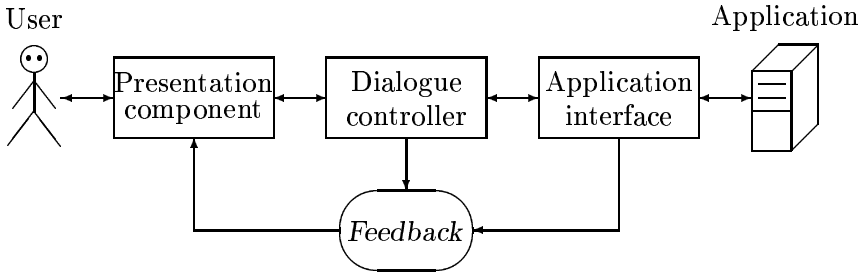


Figure 2.5: The Seeheim Model

- The *dialogue controller* implements the communication between the presentation component and the application interface. On a *syntactical* level, it describes the structure of interaction and the behaviour of the interaction objects.
- The *application interface* establishes the link between the HC interface component and the application system thus giving access to the *semantical* level, i.e., the functionality of the application. It captures the purpose of an interaction within the actual application context and passes interaction requests from either side to the other.
- The *feedback* is a direct pipe from the application interface to the presentation component. It can be used to bypass the dialogue controller when large volumes of output data are to be passed to the presentation component or when direct semantical feedback from the application system is required.

The Seeheim model is not thoroughly consistent in the focus on architectural elements only. The existence of the feedback element suggests a mix of technical with architectural concerns.

Other criticisms refer to the failure to consider the possibility of building large and complex interactive systems out of smaller components [DFAB93]. Hudson points out that the Seeheim model is inadequate for multi-threaded, direct manipulation interfaces and that a shift towards shared data (with application objects directly accessible both to the interface component and the application) is more appropriate [Hud87]. The strict separation of the interface component and the system functionality even in design is indeed a serious problem that impedes a smooth and consistent integration of the two aspects.

Heterogeneous and dynamic environments. The Seeheim model does not explicitly provide for heterogeneous environments. The handling of different social, technical, stylistic and interaction factors is not considered as a conceptual

modelling task but is completely left to the implementation of the presentation component and the dialogue control or to the implementation of the application. As a consequence, the aspect of dynamic environments where new factors can be introduced at any time is also neglected.

The Model-View-Controller Paradigm

The Model-View-Controller (MVC) paradigm has been developed for the Smalltalk programming environment [KP88]. It is illustrated in Figure 2.6 and is comprised of the following components:

- The *model* is an object or a collection of objects that represent the application domain of the HC interface. It is abstract and is used to maintain object instances.
- The *view* is responsible for the display of object instances to the user. It is either notified if changes to the object instances occur or it checks the model to detect changes.
- The *controller* manages the input from the user. It notifies the model and the view of any changes to the object instance that result from the user's actions.

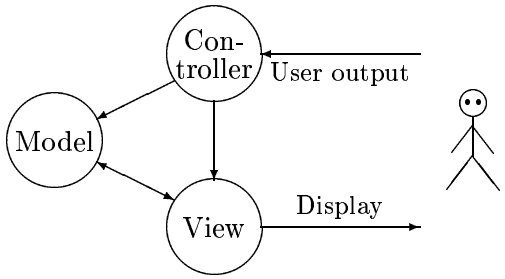


Figure 2.6: The Model-View-Controller Paradigm

The MVC paradigm quite elegantly implements the separation of application concerns and interface concerns without completely disintegrating the design aspects of overall behaviour and interactive behaviour. It also mixes, however, architectural and technical issues which is mainly due to its tight link to the Smalltalk programming environment. There is, e.g., no built-in check for the consistency of an object's model and its view. Instead, this is left to the programmer.

Heterogeneous and dynamic environments. The MVC paradigm allows to connect one model with several views so that different presentations of the same object instance are possible. This enables an adaption of the form of interaction to the HC interaction environment, but not an adaption of the flow and content of HC interaction. Adaptions of flow and content must still be dealt with by the internal application functionality.

An extension of HC interaction environments is difficult to support. It basically requires the re-engineering of the application, i.e., the identification of all interaction objects and the development of further views for each of them. As this will be a mainly manual process, consistency of presentations will be difficult to maintain throughout the entire process.

The PAC Architecture

PAC has been developed as an object-oriented model for interaction design [Cou87]. Its main constituents are presentation, abstraction and control as shown in Figure 2.7.

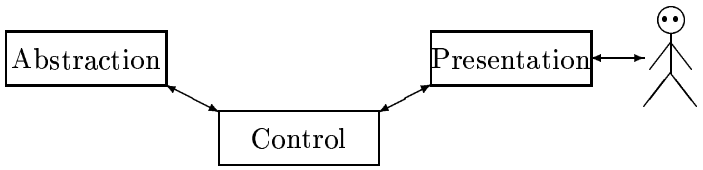


Figure 2.7: The PAC Architecture

- The *abstraction* defines the semantics of the application's interactive behaviour. It corresponds to the model component of the MVC paradigm and is used to maintain instances of application objects.
- The *presentation* describes the interactive behaviour of the application as it is perceived by the user, i.e., it provides presentations of object instances to the user and allows for manipulations on those objects.
- The *control* ensures consistency between the abstraction and the presentation components. It notifies either part if changes have occurred in the other. The control can also maintain contextual knowledge like the history of user input, explanatory information, customisation rules, etc.

In PAC, the application semantics and actual presentations are strictly separated. It is possible to change either the abstraction or the presentation without affecting the other, e.g., a change in the internal structure of an object does not

need to have an effect on its presentation and presentations can be changed while the abstraction is not.

Just as MVC, PAC can be used to build an application system in a hierarchical fashion. Indeed, the PAC architecture appears very similar to the MVC paradigm. There are, however, some vital differences [DFAB93]: *(a)* PAC unites the provision of information to and the elicitation of information from the user in the presentation component whereas MVC splits these aspects into the view and the control components. *(b)* Consistency between the abstraction and the presentation is in PAC guaranteed by the control component whereas in MVC it is left to the programmer. *(c)* PAC is not associated with a particular programming environment so that it can concentrate an the pure architectural concerns of an interactive system.

Heterogeneous and dynamic environments. Although there is no explicit reference to the possibility of assigning several presentations to one abstraction, PAC appears to be able to cater for such a situation so that it is at least theoretically possible to provide different presentations in dependence on the current interaction environment. It seems even possible that contextual information, which is managed by the control component, can be used for an automatic customisation by parameterisation. As with MVC, however, a customisation of the flow and the content of interaction is not possible. These aspects have to be dealt with by the overall system functionality.

The handling of dynamics in the interaction environment, e.g., the introduction of a new interface paradigm to be addressed, etc., involves an approach similar to that with MVC: New presentations have to be defined for each object that requires interaction. As this is mainly done manually, consistency of presentations is difficult to maintain.

The Arch Model

The Arch model [UIM92] is not so much concerned with a separation of application functionality and interactive behaviour but rather attempts to capture the nature of data that is communicated between the interface component and the other components of an interactive system. Figure 2.8 shows a graphical representation of the Arch model.

In Section 2.3 we suggested a resemblance of typical dialog system architectures to the Arch model which is due to some functional similarities of the components of both models. The Arch model contains the following components:

- The *domain specific component* controls and reorganises, i.e., manipulates and retrieves, data of the application system and performs other, applica-

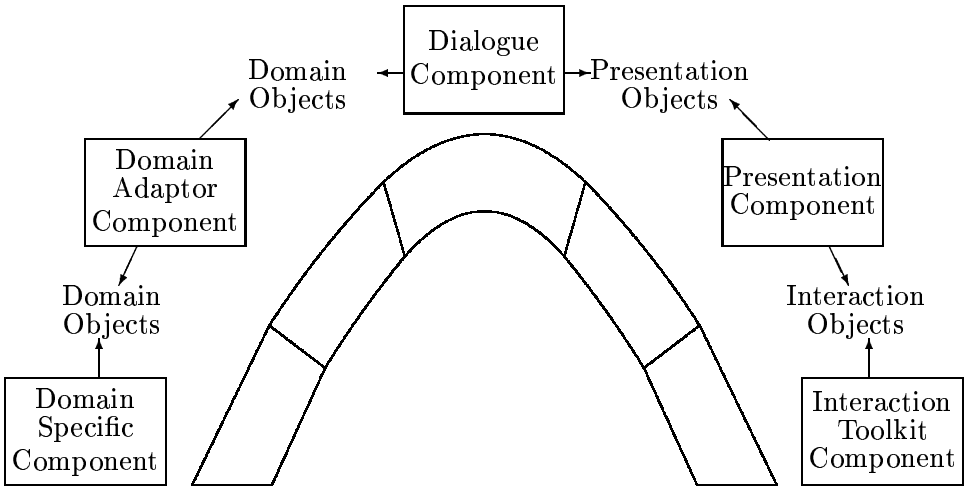


Figure 2.8: The Arch Model

tion specific functions. It provides application data in the form of domain objects.

This component corresponds to the communication with external systems in dialog systems since it gives access to the underlying functionality.

- The *interaction toolkit component* at the other end of the arch performs the actual interaction with the user using the available hardware and software, i.e., it presents interaction objects that are used for the provision of data to the user as well as for the elicitation of data from the user.

  Its equivalents in dialog systems are the speech recognition component and the speech output component.

- The *dialogue component* is responsible for the sequencing of tasks that relate both to the user and to the user-dependent portion of the application system. It translates domain objects into interaction expressed by means of toolkit-independent presentation objects and vice versa. It also guarantees consistency across multiple views and maps application specific formalisms onto interface specific formalisms.

  Together with the domain adaptor component, this component corresponds to the dialog management in dialog systems: It analyses, organises and structures dialog contributions.

- The *domain adaptor component* works as a mediator between the domain specific component and the dialogue component. It provides application specific tasks to the dialogue component that are not implemented in the

domain specific component, e.g., the ordering of list items for selection by the user. It also triggers application-initiated interaction tasks, reorganises application data and detects and gives notification of semantical errors.

- The *presentation component* is the mediator between the dialogue component and the interaction toolkit component. It translates back and forth between toolkit-independent presentation objects as provided by the dialogue component and specific constructs (e.g., graphical widgets) of the actual target interface system.

  Such an intermediate component corresponds to the language understanding component and the response generation component in dialog systems.

Each of the components of the Arch model is designed to perform a specified set of functions that are considered necessary for interactive systems. Functionality can be shifted between components so that the effects of technology changes in the future can be minimised. To support this shifting of technology the Slinky metamodel has been proposed in conjunction with the Arch model [UIM92]. The Slinky metamodel allows to shift functionality between components, thus creating different Arch models depending upon the goals of the developers, their weighting of development criteria and the type of system to be implemented.

Heterogeneous and dynamic environments. Although an adaption to different interaction environments at application system run time is not explicitly mentioned, the Arch model seems to be able to produce different interaction flows by means of the dialogue component and to produce different interaction forms (presentations) by means of the presentation component. The Arch model has, e.g., been used to create interfaces both for blind and sighted users based on the same specification [SS95]. An adaption of the interaction content, however, is only possible within the application semantics.

To support the dynamics of interaction environments, components that map between domain objects, presentation objects and interaction objects have to be exchanged by new ones. These mappings are, however, associated with concrete objects rather than the semantics of objects. Similarly to other architectures of interactive systems, new environment factors to be incorporated require, therefore, an object-based rewriting of mappings which is prone to inefficiency and above all to inconsistencies of presentations.

2.4.3 Prototypes

Several prototypical implementations for the modelling of graphical HC interaction exist. Most of these prototypes also include the automatic generation of HC interfaces based on their HC interaction models. The prototypes presented in the

following differ widely in the range of modelled interaction components as well as in the degree to which they use these models for an adaption and customisation of generated HC interaction.

Homogeneous Environments

Many prototypes assume the existence of a homogeneous interaction environment. They concentrate on the specification of flow, content and form of HC interaction alone without capturing external factors influencing HC interaction.

E.g., the **GENIUS** system uses dialog nets and entity-relationship diagrams to model the flow and content of HC interaction [BFJ96, JWZ93]. Using rules and heuristics [Wei95], window-based HC interfaces are then automatically generated. The knowledge base for this generation process includes ergonomic rules and structural information (attribute complexity, cardinalities, etc.).

JASMINUM is a somewhat less sophisticated system with regard to the aspect of HC interface generation. It uses state machines and events to model the flow of interaction [CCL94]. Application classes comprise the specification of the structure of conceptual objects, of relationships among these objects and of functions that are applied to the objects. Presentation classes that are connected to application classes specify the structure of presentation objects, relationships between them, functions that can be applied to them and functions that handle user input and output [Car94]. Thus, interaction form is fixed by presentation objects that simply are associated with application objects (interaction content). A more flexible generation of presentations on the basis of properties of the application objects (if only of a structural nature) is not possible.

Within the **PPP** CASE environment, Farshchian et al. propose to integrate the modelling of HC interfaces and application systems [FKS95]. They introduce a dialog modelling language (based on state-transition networks) for the description of the flow and content of interaction and a presentation language for specifying the form of interaction. Both languages are intended to be easily integrated with other languages used for application modelling in the CASE environment. Although it is stated that this approach is based on the Seeheim model ([PH85], cf. Section 2.4.2), it uses so-called domain objects which are shared between the application and the HC interface. This has the advantage of easy communication between application data objects and interface data objects but bypasses the application interface—a fact which does not conform to the Seeheim model.

Customisation of HC interaction. The prototypes presented above are aimed at homogeneous environments only and thus do not consider and cater for a customisation and extension of HC interaction at all.

And yet, a customisation of interaction in heterogeneous environments seems possible for the form of interaction either by the redefinition of mapping rules in the case of GENIUS or by the substitution of attached presentations in the case of JASMINUM and PPP. An extension by new presentations should be most efficiently possible in the case of GENIUS since it can be performed independently from the actual model simply by the replacement of mapping rules.

Heterogeneous Environments

A wide range of prototypes exist that—in contrast to approaches that assume homogeneous interaction environments—take external factors into account.

In particular, model-based approaches like **HUMANOID** [SLN92], **MOBILE** [PCOM99], **TADEUS** [SE95] and **TRIDENT** [BHLV95] gather information on the user to be addressed and the tasks to be fulfilled. A customisation of interaction aspects like the flow of interaction or the form of interaction based on this information is then possible but not always implemented. Customisation of interaction content is hardly anywhere considered.

The **NOODL** language adopts a unifying view on application objects by associating data objects and interface objects [MKB95]. Data objects describe the structure of database objects and possible operations on them; interface objects then provide features relating to the interactive behaviour and visual representation of these objects. Triggers can modify values of interface object properties so that a customisation at system run-time is possible. The NOODL approach has been taken a step further in the provision of a framework for visual interfaces to databases [MKB96]. In this framework not only heterogeneity with regard to the user but also with regard to the database is considered. A visualisation and an interaction component interact both with different user types and database systems. The visualisation component is responsible for providing information to the user. It distinguishes between the aspects of contents to be presented (referent), of symbols used for visualisation (metaphor) and of positioning different interface components (layout). The interaction component deals with the acceptance of user input and comprises the aspects of activities to be performed (intention), of mechanisms offered for performing these activities (medium) and of results of activities on any component of the entire system (effect). Both components thus offer different layers of abstraction for the modelling of interaction content and form.

Different interaction paradigms are also provided for in the work of **Paton** et al. [PDDJ96]. An internal representation of application objects (describing the structure of objects and the functions on them) is linked to different pre-defined interface representations. This approach also promises support for the adaption of interfaces to new functionality of database systems but offers little to back this claim.

Multi-paradigmatic access to databases is also the theme of the work of **Catarci** et al. [CCM+96, CCC+96]. In their approach, the underlying database is modelled by a unifying Graph Model [CT95]. Application objects involved in interaction are then associated with different pre-defined visualisations (e.g., diagrammatic, iconic and virtual reality). These visualisations are generated by algorithms on the basis of the graph model describing the database, i.e., the nodes of the graph and the constraints associated to the graph are mapped onto visual presentations and relations between them. Depending on their characteristics, users are offered one of these interaction styles or can select one according to their own tastes. The modelling of interaction flow is only considered in terms of query formulation.

Teallach is another model-based approach to platform-independent interface specification and to that end uses the platform-crossing abilities of Java [GMP+98]. Flow, content and form of interaction are captured in a task, a domain and a presentation model. User characteristics are contained in a user model. The presentation model consists of two levels: On an abstract level it describes a generic interface that represents the corresponding task model. On a concrete level it uses Java Beans to realise a concrete interface instance.

Customisation of HC interaction. All the presented systems allow for a customisation of the form of interaction either by a redefinition of mapping rules or by the provision of several presentation objects for one application object. A customisation of the flow and the content of interaction is not considered but is left to the general functionality of the system.

An extension of customisation to face new interaction situations seems to be best supported in the prototype described by Catarci et al. [CCM+96, CCC+96]. Just as is the case with GENIUS, the use of rules that define general mappings between concepts of the interaction model and concrete presentations allows to define new presentations independently of the model; a re-engineering of the application specification is not necessarily required. And yet, the ease of redefining rules depends on the quality of implementation, on criteria such as modularity that support reuse. Quality of implementation, however, cannot be enforced by any model or framework. It will always be a decisive criterion for the ease of system extension and modification.

Heterogeneous Environments – Extensibility

The unifying characteristic of the above approaches is that presentations are to a large extent pre-defined and that very little flexibility for customisation is offered. Some approaches use structural properties of the objects to be presented to generate presentations [JWZ93, CCC+96] or provide triggers to modify values of interface properties [MKB95], but they still fail to exploit the semantics of

the information to be exchanged and do not always allow for a customisation of interaction without the need to re-engineer the entire application.

The work of **Díaz** et al. concerns the support of dynamic displays using active rules [DJPQ94]. Interaction content and flow are modelled by means of objects (and object classes, respectively). Objects are directly associated with representations as in many other approaches. In this case, however, active rules are used to notify the presentation component not only of modifications of concrete objects but also of modifications of object classes. Hence, to the degree where structural modifications of object classes can be foreseen (e.g., attribute extensions, etc.), presentations can be adopted independently from the concrete application.

The **Iconographer** system has been developed for the rapid prototyping of iconic representations [GWD90, DW91]. Although it is concerned with specifying the form of interaction only and assumes the specification of interaction flow and content within the underlying application system, it offers an interesting and flexible method of controlling visual presentations. In the Iconographer system it is not entire objects or subobjects that are linked to presentations, but *properties* (attributes) of objects. E.g., if an object file is linked to a particular icon, the attribute size can be used to control the 'thickness' of that icon. As in most other systems, such links are defined in dependence on particular objects; there are, however, two issues that make this approach interesting to interaction in heterogeneous environments: *(a)* The semantics of objects are at least partially considered. Individual object attributes and their concrete values instead of complex object structures are mapped to presentations. *(b)* The concept of attribute types is considered. Certain types are (by default) not compatible, e.g., the types file name and icon colour, although even here the designer may define correspondences. For compatible types, however, default mappings are provided that convert between domain ranges. This principle of type mappings could well be extended to provide mappings independently from particular objects. The Iconographer system has later been extended to the Representer system that among other issues provides more efficient means to specify the mappings between attributes and presentations [Gra97].

An interesting recent example for the application of the Iconographer concepts is the SOMLib project [RM99]. The SOMLib project is a digital library prototype; one of its components, the libViewer, is responsible for a visualisation of the individual documents [RB99]. Among other features, it allows the definition of mappings between document meta data and intuitive graphic metaphors which will be attached to the icons used. Some examples of such mappings are: age of document maps onto 'shininess' of binding, publisher maps onto logo on icon, time of last reference maps onto the amount to which an icon is pushed to the back of the bookshelf, etc. The redefinition of these mappings is possible so that some degree of flexibility of presentations is achieved.

Customisation of HC interaction. Both the prototype described by Díaz et al. and the Iconographer/Representer system are suited for the customisation of some components of interaction in heterogeneous environments: the Díaz system considers a customisation of form and content, the Iconographer system of form only.

When considering the support of dynamic environments, for the Díaz system the same is true as for Catarci et al.: The ease of redefining mapping rules depends on the quality of rule implementation. In the Iconographer system, rules always map individual modelling concepts onto individual presentational aspects. This reduces the complexity of mapping rules and a redefinition is at least facilitated. However, rule designers and implementers are still responsible for a sufficient modularisation and programming quality to allow for an easy redefinition.

2.5 Motivations Gained and Further Requirements

In Sections 2.1 to 2.4 we discussed a number of approaches to the description and the support of interaction in general and HC interaction in particular. Each of these approaches focuses on different aspects like the influences on interaction or the modelling and generation of interaction, which is also due to the differing backgrounds and intentions of each approach.

The aim of this work is to unify these aspects, i.e., to provide a framework for a unified consideration of these aspects. The following section will, therefore, discuss which motivation for such a framework we can draw from the previously presented approaches and which further requirements need to be addressed.

The influence of the interaction environment. A frequently occurring issue is the influence of different aspects of the interaction environment on the way interaction is performed. Lasswell emphasises the importance of the receiver, the sender and the channel of a message; Shannon is concerned with the (technical) properties of the interaction channel too; Bühler has pointed out the effects of conventions for sign usage, of the sender's intentions and the receiver's expectations on the construction and use of signs. Dialog systems attach much importance to the flexibility of their dialog management and response generation components. Finally, models as well as prototypes of graphical interface systems acknowledge the fact that different user types and interface types have to be catered for.

Dynamics of interaction environments. Interaction environments change constantly. This is one of the facts that inspired interactive computing.

Because interaction environments are dynamic it is indispensable *(a)* to provide means for HC interaction to support an exchange of previously unknown information with the user and *(b)* to provide a customisation of the means of interaction to meet new requirements posed by various environmental influences. It is the second of these aspects that motivates this work.

Further requirements. Besides motivating the support of HC interaction, many of the discussed approaches also provide methods and means for such a support. The issues that are of importance are the modelling of HC interaction, the use of these models to automatically create customised HC interaction and the possibility to efficiently respond to new requirements of the interaction environment:

Modelling HC interaction: Methods for the modelling of all three components of interaction (flow, content and form) exist (cf. Section 2.4.1). Not all of these methods do, however, provide a level of abstraction that is sufficient for a customisation of interaction.

This is especially true for the *form* of interaction. Interaction form often is either modelled by means of *(a)* presentation objects—which makes it difficult to respond to new requirements in the interaction environment—or *(b)* is derived based on structural information on the interaction content (e.g., GENIUS, Catarci)—which is not enough to generate very sophisticated interfaces—or *(c)* is created based on links between properties of application objects and presentational properties (e.g., Iconographer)—since these links are still related to specific application objects, this approach is not general enough.

In the remainder of this work we will provide means for an abstract and explicit modelling of interaction form—while we rely on traditional means for the modelling of interaction flow (an enhanced Petri net variant) and interaction content (data type definitions). The use of semantical modelling constructs as motivated by many dialog systems will enhance structure-based translations (GENIUS, Catarci) on a more general level than would be possible by means of object-based property mappings (Iconographer). Semantical constructs together with the used Petri net variant and the employed type model ensure a complete modelling of all the aspects relevant for interaction.

Customising HC interaction: The approaches that were discussed in the previous sections mostly focus on the customisation of interaction form (by the provision of several sets of presentations objects or of different rule sets). Only dialog systems and the Arch model pay attention to the customisation

of flow and content as well. A methodical consideration, however, is still lacking.

In the following, we will present a model and mechanisms that allow for a thourough customisation of all components of interaction. The modelling of all components will allow the specification of conditions and semantical properties that determine the applicability of the modelled concepts in specific interaction situations. Moreover, the available concept of various rule sets is again applied—this time to address different interaction media.

Responding to new requirements: The Iconographer system has an efficient mechanism for the customisation of interaction that also allows to efficiently accommodate extensions of the interaction environment: These can be catered for by the redefinition or the extension of mapping rules.

In the remainder of this work, we use this mechanism again (in the form of presentation rules) but it can now be applied much more efficiently due to the existence of general abstract and explicit constructs for the modelling of HC interaction. These modelling constructs enable designers to concentrate on the redefinition of mapping rules alone without having to decode the intended meaning and effects of the constructs used for application modelling.

Part II

The ACE Framework for Abstract Modelling and Automatic Customisation of Human-Computer Interaction

Many information services target a heterogeneous and dynamic audience. When developing HC interaction for such services, we face the challenge of being as cooperative as possible in as many situations as possible. This means that the content of interaction (the information to be exchanged), the flow (the navigation patterns) and the form (the concrete presentation) must be customised to the requirements of individual interaction situations. Heterogeneous and dynamic interaction environments call for an approach to the development of HC interaction that pays full attention to such a diversity. HC interaction must be customised to the encountered interaction situation if interaction between the information system and its human users is to be successful and efficient.

The need for such a flexibility also requires changes in the approaches to the conceptual modelling of HC interaction. Flexibility—especially with regard to unexpected changes in the interaction environment—cannot be achieved on the basis of a design that has already fixed upon particular properties of an anticipated environment.

The following chapters present a framework—the ACE framework—for the abstract conceptual modelling of HC interaction that serves as the basis for the generation of flexible and customised HC interaction. They also illustrate how this framework can be applied.

Chapter 3 outlines the ACE framework whose principles are *abstraction* of modelling, *completeness* of all relevant aspects in modelling and generation and *explicitness* of modelling and translation mechanisms. We describe the dimensions of the framework and their role within the functioning of the framework.

The modelling of HC interaction is one of the aspects covered by the ACE framework. Chapter 4 presents an overview of the HC interaction modelling concepts proposed. These concepts have been motivated by traditional HC interaction design and other influences like linguistics, screenplay writing and graphical design.

In Chapter 5 we illustrate the process of HC interaction modelling. In particular, we describe the abstraction layer Co-Design model and the incorporation of HC interaction modelling into this model.

Finally, Chapter 6 shows how an abstract model of HC interaction can be exploited for the automatic generation of HC interaction. We point out the principles of HC interaction generation, propose an architecture and present a case study that illustrates which effects the different properties of interaction environments can have on the generated HC interaction.

Chapter 3

The ACE Framework for Human-Computer Interaction

The support of HC interaction for heterogeneous and dynamic environments also requires changes in the approach to HC interaction *development*. In the following, we will point out what is required of a development approach targeting heterogeneous and dynamic interaction environments. We will then present the ACE framework that aims to fulfill these criteria. The ACE framework is described, it is explain how it is to be put to use and what its benefits are.

A detailed discussion of HC interaction modelling and generation within the ACE framework is provided in Chapters 4 to 6. Chapters 7 to 9 then describe a prototype that has been used to adopt the principles of the ACE framework.

3.1 Principles of Human-Computer Interaction Development

The existence of heterogeneous interaction environments involving different interaction partners (users) to be addressed, different target systems and communication channels to be used and different styles to be created has put new challenges to the way HC interaction is realised and—as a prerequisite—modelled. HC interaction must provide *customisability* which means that different interaction situations must be catered to; it also must provide *extensibility* which means that an incorporation of previously unexpected interaction situations should be facilitated. Both customisability and extensibility contribute to the *usability* of a system [Nie93]. Usability is not only a matter of offering 'good' interaction to users but also of offering *different* interaction to different users in different situations—interaction that is customised to the respective requirements.

Customisability and extensibility can be supported by adhering to the three methodical principles of *abstraction*, *completeness* and *explicitness*:

Abstraction means that the possibility for an abstract description of HC interaction is provided which does not pay attention to any details of eventual interaction situations—neither to interface systems to be addressed nor to concrete appearances, ideally not even to user types. Instead an abstract model is to be devised that can serve as a starting point for automatic HC interaction generation in all possible interaction situations. To give a simple example, characterising an interaction entity as little important, could allow automatic generation mechanisms to drop this entity in case of size restrictions of the interaction channel or the interaction medium and/or to find various concrete presentations like small font, display at the bottom of a screen, provision only after follow-up questions, etc.—just what is most appropriate in the current situation.

Abstraction is not an absolute notion. Interaction developers can (and indeed often will) restrict the degree of abstraction depending on the requirements of the particular application context. Abstraction is reduced by referring to properties of the environment directly in the HC interaction model. This involves the consequence, however, that changes of the environment may also require changes of the model which impedes the extensibility of the design. However, the reasoning that would be required to generate concrete interaction based on completely abstract models would often (at least with given technologies) appear too complex and thus not feasible.

Completeness means a thourough description of *all* the aspects that will guide a concretisation of an abstract interaction model. This includes two issues: firstly, the model itself, i.e., a complete description of the semantics of HC interaction (cf. Chapter 4); secondly, completeness includes the environmental factors that will *influence* how the model is transformed into physical representations.

Again, completeness is not an absolute notion. Some possible aspects of the model or the environment might not be relevant for a concretisation, others might be considered too difficult to specify or to extract, yet others would be too difficult to utilise. Interaction developers are free to realise completeness of HC interaction modelling according to the actual requirements and possibilities of the application.

Explicitness means that the effects of aspects (from the interaction model as well as from the environment) on the concretisation of interaction should be specified as openly as possible and should not be hidden inside complex translation mechanisms.

To realise explicitness lies to a large extent in the responsibility of program-

mers or in the provision of tools that realise an automatic management of the specified effects.

Customisability is supported by the principles of abstraction and completeness. These ensure the existence of a common model of HC interaction based on which HC interaction generation for different interaction situations can be realised. *Extensibility* is enabled by the principle of explicitness since an open specification of the effects of the modelled aspects facilitates their redefinition.

3.2 The ACE Framework

The following section describes the structure of the ACE framework and its utilisation for the modelling and the generation (concretisation) of customised HC interaction. The ACE framework contains the two dimensions of *scope* and *abstractness*. Whereas the scope of HC interaction describes the individual components that constitute interaction, abstractness captures the process of 'stepping down' from abstract conceptions of what is to be communicated with regard to each component to concrete presentations of those conceptions, i.e., the process of materialising HC interaction into a physical form (generation). The structure of the ACE framework is shown in Figure 3.1.

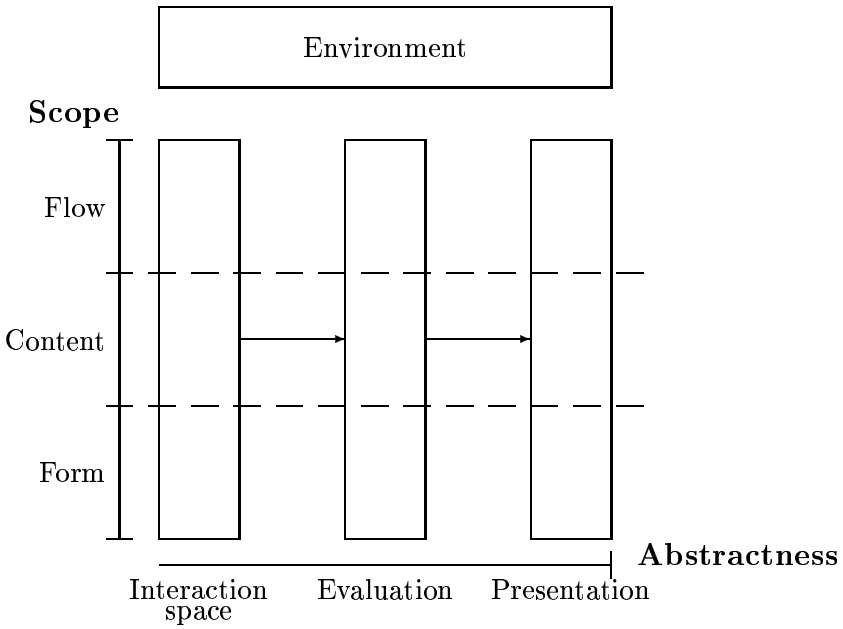


Figure 3.1: The ACE Framework

The environment. An important aspect of the ACE framework is information on the interaction environment since it is to be used for a customisation of interaction. Environmental information includes social, technical, stylistic and interaction factors (cf. Section 4.6).

The scope dimension. The scope dimension reflects the components of HC interaction which were already introduced in Section 1.2.2: interaction flow, interaction content and interaction form.

Interaction flow describes the activities involved in interaction and their sequencing. Sequencing can be due to causal relations or to mere temporal relations (out of ergonomic considerations only). It is possible to introduce normative aspects to describe whether and under which conditions activities are allowed.

Based on the properties and requirements of the current interaction situation, certain activities will eventually be selected for an execution and a concrete temporal ordering of them will be realised.

Interaction content specifies the information involved in HC interaction. It describes interaction entities (data) that are associated with the individual activities. Interaction entities can be associated with activities as input data, i.e., data to be provided to the user, or output data, i.e., data to be extracted from the user. Again, normative aspects can be introduced.

Eventually, interaction entities will be selected and instantiated with concrete values. For input data, means for their presentation to the user have then to be found while for output data mechanisms for accepting user input have to be provided.

Interaction form refers to the presentational semantics of the information involved in HC interaction. This can include the prominence of information, various application-dependent properties that are to be visualised or articulated, relations between data items (e.g., cohesion, sequences) or usage properties of information (e.g., warnings, alerts).

According to the current interaction situation, presentational semantics will finally be selected and instantiated and then be translated into physical presentations. They have a great influence on the appearance of the means chosen for the presentation of interaction content and on their arrangement.

The abstractness dimension. Whereas the scope dimension can be used to capture the more or less static structure of HC interaction at a given point of development, the abstractness dimension describes the dynamics during the

process of HC interaction concretisation or materialisation. It involves the phases of defining the interaction space, of evaluation and of presentation. It is during the materialisation of HC interaction that a customisation of interaction according to the current interaction situation takes place.

The interaction space corresponds to an abstract conceptual model of what to interact about, of what is supposed to be going on—thus describing the *'What'* of interaction. It includes all possible interaction activities and interaction entities, their relations and the description of their meaning. Ideally, specific interaction situations are not considered.

The evaluation then decides on the *'Which'*. Based on normative aspects, it selects those activities, interaction entities, dependences and semantic properties that apply in a specific interaction situation. It is also responsible for the instantiation of interaction entities and semantic properties according to the situation encountered.

The presentation of interaction settles the final question of *how* to interact. It decides on the eventual constructs chosen to present information, navigation structures, etc., as well as on the visual/auditive properties of those constructs and their spatial and temporal arrangement.

During the process of materialising HC interaction (abstractness dimension) each component of HC interaction (scope dimension) has to be dealt with. Table 3.1 shows in some more detail how the individual components are captured and implemented on different levels of abstractness, thus describing how the instances of the ACE framework are expected to be filled.

HC interaction modelling within the ACE framework. The modelling of HC interaction involves the description of the interaction space as well as the interaction environment (cf. Figure 3.2). Since concrete presentations are supposed not to play a role during the modelling of HC interaction, the modelling of the interaction space concentrates on an abstract description of *possible* interaction flows, contents and forms. Information on the interaction environment describes social, technical, stylistic and interaction factors that have been declared relevant for an eventual concretisation of interaction.

Section 4 provides a collection of constructs for an abstract modelling of HC interaction and discusses aspects of environment modelling

HC interaction generation within the ACE framework. It is along the abstractness level that a materialisation of HC interaction, i.e., its generation, takes place: Whereas the interaction space consists of an abstract model of all

Abstract-ness / Scope	Interaction space	Evaluation	Presentation
Flow	Characterisation of activities, necessary causal and temporal dependences between activities, normative aspects	Selection of relevant activities and dependences based on interaction situation	Implementation of task sequences, navigation structures, etc.
Content	Interaction entities (participating input and output data), characterisation, normative aspects	Selection and instantiation of data entities based on interaction situation	Invocation of constructs available on target system for data provision and user input
Form	Semantic description of participating data entities, normative aspects	Selection and instantiation of semantic properties based on interaction situation	Visual/auditive properties and arrangement of constructs used for the conveyance of interaction content

Table 3.1: Dimensions of Interaction

required and possibly relevant HC interaction issues, this collection will be condensed during the *evaluation* phase to those issues only that are relevant in the interaction situation presently encountered. Furthermore, where required, values for input data or semantic properties are instantiated. In the *presentation* phase, a physical shape is given to all selected and instantiated concepts of the original HC interaction model. These activities are illustrated in Figure 3.3.

The evaluation and presentation phases require an analysis of the current interaction situation. This can involve the consultation of various models, the checking of environment variables of the operating system, reasoning on available information and/or direct questions to the user. The information thus acquired controls the applicability of the evaluation and presentation rules on the one hand and their instantiation on the other hand.

When moving towards an *evaluation* all components of the HC interaction model, i.e., the interaction space, are involved.

- With regard to the interaction *flow*, it is determined which activities are selected and which dependences between them are to be realised. An activ-

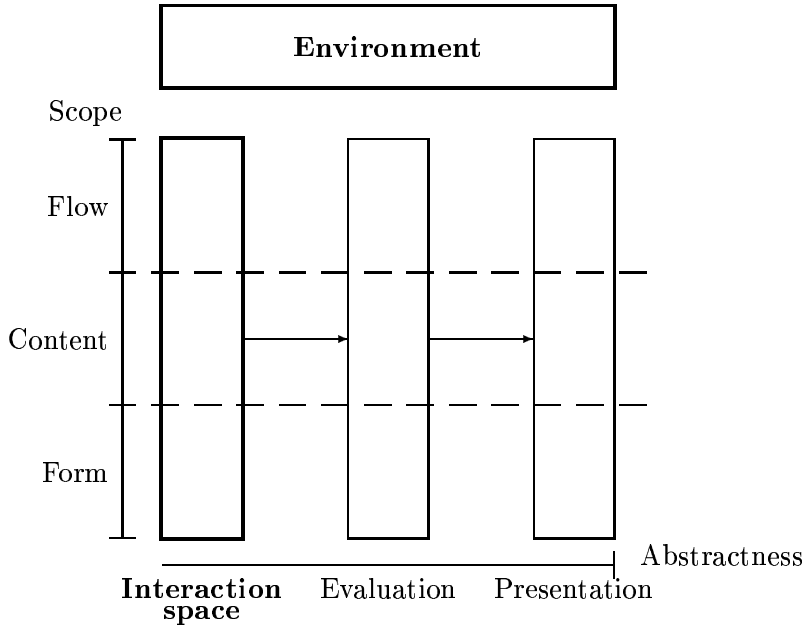


Figure 3.2: Modelling HC Interaction in the ACE Framework

ity could, e.g., have been specified as a 'booking activity' and would only be selected if the respective user is known to be interested in performing bookings. Temporal dependences (describing scenarios) can be specified for information seeking or for browsing purposes and can then be selected according to the user's current purpose. It is also possible to realise different 'lines of persuasion' to different users so that product offers are better accepted.

- Furthermore, the interaction *content* is decided upon. E.g., an interaction activity could have different interaction entities as a starting point: one including help information for the user and another plain entity. The additional help information could be selected for inexperienced or new users whereas experienced users would probably prefer plain information.

  Similarly, the instantiation values can be selected in dependence on the current user, targeting higher-priced offers to customers that are known to be price-insensitive or budget offers to students.

- Finally, semantic properties that influence the *form* of interaction can be selected. Not all semantic properties are of interest to all users; some users might want to have price information highlighted, others do not. During evaluation those semantic properties are selected that are of interest in a particular interaction situation. Additionally, values of semantic properties

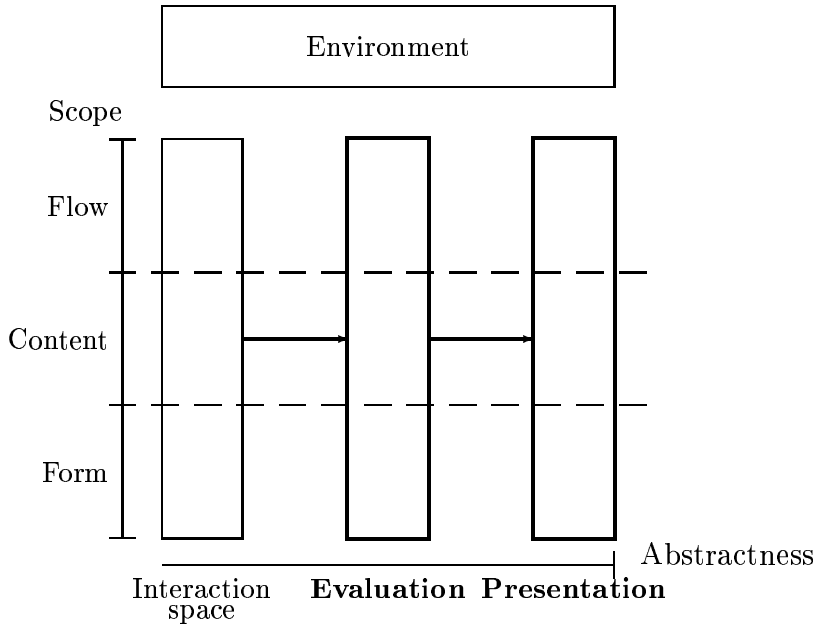


Figure 3.3: Generating HC Interaction in the ACE Framework

(e.g., the degree of highlighting) can be determined.

The result of the evaluation is a 'message' to be communicated to the user: which interaction activity is to be performed, which content is to be conveyed, which semantical properties are of relevance and which further activities might possibly be offered to the user. This message now requires a format, a physical shape which is given to it in the eventual presentation.

During *presentation*, the eventual appearance of interaction is decided upon. Just as the evaluation, this requires a thourough analysis of the entire interaction environment: of the constructs available on the targeted interface system, of bandwidth restrictions, user preferences, possible physical handicaps, etc. Different interaction situations are likely to require very different presentations of the same message to be communicated.

- In terms of interaction *flow*, the presentation decides in which manner dependences between interaction activities are to be realised: by navigation structures (which requires input by the user by clicking on buttons, hyper links, selecting menu entries, etc.), fixed tasks sequences, etc.

- Interaction *content* is mapped onto constructs as available on the current interface system and as suitable to the preferences and the cultural background of the current interaction partner (user). The mapping of interaction

content can be based on type mappings such as mappings from STRING onto text fields.

- Finally, interaction *form*, i.e., the semantic properties of interaction entities, is used to control the arrangement of the selected constructs and to decide on visual/auditive properties like colour, size, voice volume, etc.

3.3 The Abilities of the ACE Framework

The ACE framework enables the support of the principles of HC interaction development for heterogeneous and dynamic environments (cf. Section 3.1):

- It provides for an *abstract modelling* of HC interaction that can serve as a starting point for different concretisations.
- It supports *completeness*, i.e., it models and considers customisability for all components of HC interaction, not only for the form of interaction.
- The *explicit* modelling of interaction components and of the factors that influence the concretisation of interaction, also makes the explicit definition of the mechanisms used for customisation possible.

The abstract modelling of HC interaction within the ACE framework is described in Chapter 4. Based on this discussion, Chapter 6 explores the mechanisms for a concretisation, i.e., an automatic generation, of HC interaction. Following, we use a few examples as illustrations of the abilities of the ACE framework with regard to a customisation of interaction to different interaction situations:

Customisation of flow:

- *Normative aspects* introduced in the abstract model can determine in which interaction situations activities are to be executed or not.
- *Sequence relations* can be analysed to prioritise the execution of certain activities over the execution of others. (A mechanism for the classification of activities into priority clusters has been developed in relation to database design activities [Lew96].)
- The physical *realisation* of interaction flows can result in different mechanisms such as the provision of menus, buttons, hypertext structures, etc. There is also a choice between user-initiated and system-initiated flow control.

Customisation of content:

- *Normative aspects* can determine which interaction entities are to be consumed and produced, respectively, by activities, i.e., which data is provided to the user and thus required to start the activity and which data is to be extracted thus ending the activity. Among other things (such as the incorporation of additional help information) this mechanism can be used to choose between different types of data, e.g., text vs. image data, depending on the capabilities of the target interface system and platform.
- *Parameters for the instantiation* of data entities can perform modifications of the data retrieved from the underlying database or state conditions for the retrieval itself so that in principle the actual content provided to the user can differ (although this can, of course, raise some ethical questions).
- The *presentation* of content is above all dependent on the concrete means supplied by the targeted interface system. Is is, however, often possible to choose between different mechanisms, e.g., providing STRING data as text or as spoken language in case an interface system offers both possibilities. Moreover, it is conceivable to provide different presentations for complex data that has been modelled in a canonical form, e.g., a path description consisting of a number of streets, turning points and landmarks could well be translated into a textual representation ('Walk along *Virchowstraße*, then turn left into *Braillestraße*...) or into a graphical representation, e.g., a map.

Customisation of form:

- Again, *normative aspects* can determine which characteristics and relations, that shape the eventual appearance of interaction, are relevant in a specific situation, e.g., whether price information is marked important or not, etc.
- The *instantiation* of values to characteristics can equally be made dependent on the encountered situation, e.g., to mark price information as *very* important for students and 'just' important for other customers.
- How to *represent* these characteristics and relations in a concrete shape is also open to customisation. The importance characteristic could, e.g., be visualised by a certain font size, colour, blinking text, etc.

Chapter 4

The Modelling of Human-Computer Interaction

One of the aspects provided for by the ACE framework is an abstract and complete modelling of HC interaction. The modelling of HC interaction corresponds to the abstractness level of interaction space; it consists of a collection of all possible and relevant properties of interaction flow, content and form. Moreover, properties of the interaction environment that are of potential influence on the concretisation of HC interaction need to be identified and mechanisms for their representation must be provided.

In the following, we present a number of abstract constructs for the definition of the interaction space of a high-level information system. These constructs are motivated by existing computer science research as well as by concepts from linguistics and graphical design. Section 4.1 provides an overview of the design model proposed thus serving as an orientation to Sections 4.2 to 4.4 which describe the individual modelling concepts. Section 4.5 puts forth how exactly these modelling constructs map onto the individual interaction components, while Section 4.6 is dedicated to the modelling of the interaction environment.

Based on this chapter, approaches to the support of HC interaction modelling are then presented in Section 5 and methods for HC interaction generation on the basis of these abstract models are discussed in Section 6.

4.1 Model Overview

In Section 1.5 we have defined an interaction model as a 5-tuple (P, E, T, E_0, E_F) with

- P being the set of interaction partners p,
- E being the set of interaction entities e,

- T being the set of transformations (activities) t transforming entities,
- E_0 being the set of initial entity sets and
- E_F being the set of possible final entity sets.

Based on an interaction model, different instances, i.e., concrete interactions, are possible.

In the following sections, we will establish constructs to model HC interaction. These constructs relate to the definition of interaction we use in the following way:

- Different human interaction partners are described by *actors* and their right to perform certain transformations. The information system as an interaction partner is not modelled explicitly; it simply is responsible for performing any computational process. Receiving interaction partners are neither explicitly modelled; a receiving interaction partner is any human user whose access rights allow to perform another transformation that consumes the produced interaction entity as starting entity.
- Entities are modelled as *data*.
- Transformations are modelled by computational or interaction *processes*. Together with entities they build a bipartite graph thus describing the overall structure of interaction. The overall structure is further controlled by various process characteristics and relations.
- A concrete interaction can *start* with any set of entities as long as these are instantiated. Entities that have been used to start interaction processes can be kept in the interaction part of the environment model.
- Often, *final entities* are all entities that can be reached from the current point of interaction. In some cases, however, the user is required to perform transformations that lead to a certain entity and is not allowed to quit during that time (e.g., after having agreed to a booking, all the required information to allow for a payment must be given). In that case, this information is held in the interaction part of the environment model and an interaction flow leading to exactly the given point must be modelled and created. Exceptions such as a break-down of the system must be handled by the control part of the system.

In the following, we will introduce a language for abstract HC interaction modelling which incorporates these concepts. A meta model of this language is shown in Figure 4.1 in HERM notation [Tha00]. The proposed language has the following features:

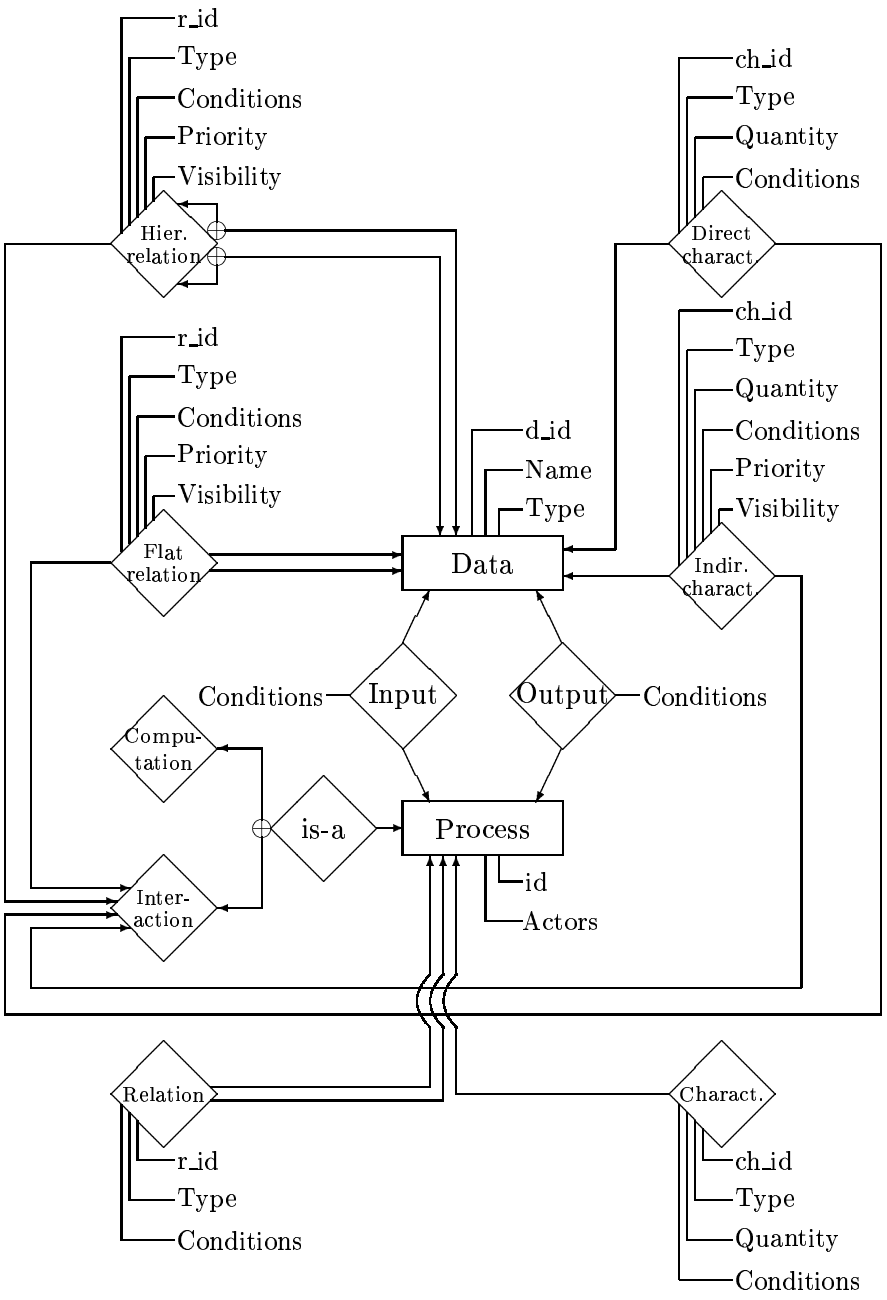


Figure 4.1: Meta Model for the Modelling of HC Interaction

- *Data* are modelled by means of a relatively simple type model which contains certain base types and complex types. The substitution of this type model by a more complex type model is, of course, possible.

- *Processes* are modelled as transformations between data. The connection of process elements and data elements thus creates a graph: a bipartite hypergraph [CLS99b]. Bipartite hypergraphs are a Petri net variant and have the same expressiveness as Petri nets. Due to more flexible semantics to describe connections between data elements and process elements (e.g., conditions to specify data to be consumed or produced), they avoid, however, the 'explosion' of Petri nets in more complex applications.

  Data serve as input and as output to processes. Characteristics and relations of processes are an additional modelling means to control the execution of processes.

  The detailed specification of computational processes is part of the behaviour design aspect and thus is not considered here. Interaction processes result in the generation and the presentation of interaction objects (cf. Section 6). Thus their modelling encompasses the modelling of the participating data (by means of data types) and the modelling of characteristics and relations of the participating data.

- *Characteristics and relations* allow to model a number of attributes of data or processes. Since they control the way processes are initiated and performed, both are part of the modelling of processes. Data characteristics and relations are part of the modelling of interaction processes only; they are to interaction processes what algorithms are to computational processes: They determine their physical results, i.e., in this case, their appearance to the human user.

In the following sections, we present and motivate a detailed description of the language constructs proposed for the conceptual modelling of HC interaction for high-level information systems in heterogeneous and dynamic environments. An exemplary syntax is partially introduced during this discussion. The complete syntax can be found in Appendix B. Occasionally a graphical representation of that syntax will be used which is to a certain degree more easily readable.

4.2 Data

Data correspond to interaction entities. They are used to capture the information that is to be provided to and/or to be extracted from the user as well as the information that plays a role in interaction-related data processing.

For data modelling we use the following syntax:

```
⟨data⟩     ::= ⟨d_id⟩: ⟨name⟩: ⟨d_type⟩
               | NUL
⟨d_id⟩     ::= D#⟨n_list⟩
⟨n_list⟩   ::= ⟨number⟩
               | ⟨number⟩.⟨n_list⟩
⟨name⟩     ::= ⟨word⟩
⟨d_type⟩   ::= ⟨basic⟩
               | ⟨complex⟩
⟨d_list⟩   ::= ⟨data⟩
               | ⟨data⟩, ⟨d_list⟩
⟨basic⟩    ::= STRING | CHAR | NAT | REAL | BOOL | DATE
⟨complex⟩  ::= SET (⟨data⟩)
               | LIST (⟨data⟩)
               | RECORD (⟨d_list⟩)
```

Data are described by an identifier, a name and a type. We use a relatively simple type model of base types (String, Char, Nat, Real, Bool and Date) and complex types (Set, List and Record). Principally, the use of a more complex type model or an incorporation of semi-structured or unstructured data is possible and often required, but it is not considered in this work. In any case, each data type must be translatable to the underlying database, e.g., by means of view definitions.

Within complex data types (also called data hierarchies in the following) each data node must still be identifiable to enable the assignment of semantic characteristics and relations (cf. Section 4.4). The assignment of an identifier to the root node of a data hierarchy makes it possible to use the same data type in different roles, i.e., entities of HC interaction. The use of compound identifiers (e.g., D#1.2.4) is recommended but not required to allow for an easy identification of data nodes.

The following example shows the specification of an application form in the provided syntax. Figure 4.2 depicts a corresponding graphical representation.

Example 4.1: A type definition

```
D#1:
 Hotel_reservation:
 RECORD (D#1.1:
          Customer:
          RECORD (D#1.1.1:
                    Name:
                    RECORD (D#1.1.1.1: First_Name: STRING,
                            D#1.1.1.2: Last_Name: STRING),
                  D#1.1.2:
                    Address:
                    RECORD (...),
                  D#1.1.3:
                    Phone: NAT),
         D#1.2:
          Rooms:
          SET (D#1.2.1:
                Room:
                RECORD (D#1.2.1.1: Number: NAT,
                        D#1.2.1.2: Price: REAL,
                        D#1.2.1.3: Period: RECORD (Begin: DATE,
                                                   End: DATE))))
```

■

4.3 Interactions and Computations

Between data, transformations take place, either by HC interaction or by algorithmic computation. Both types of transformation processes have an identifier and are associated with input data, output data and actors.

Interaction processes are points where the information system requires the human user to input data, to confirm data, to select data, etc., or where data simply is to be provided to the user.

Computational processes are used to process data for the support of interaction processes.

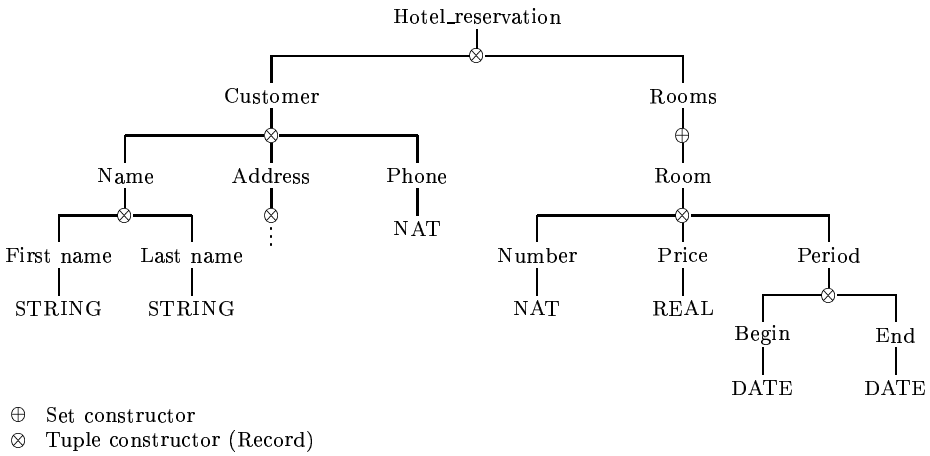


Figure 4.2: Interaction Data

A computational process transforming data is easy to interpret: it consumes its input data and produces some output data. In the case of the null entity serving as input data (missing input data), it simply retrieves data from the database; in the case of the null entity serving as its output data (missing output data), it simply stores data.

An interaction process works according to the same principle, but here the 'processing unit' is the user. Input data of an interaction process is consumed by the user, i.e., it is provided to the user by the system; output data is produced by the user—typically on the basis of previously provided input data—and is given back to the system. Missing input data means that the user only enters data without other data having been displayed as an input; missing output data means that the user only consumes (i.e., takes notice of) data without producing any output.

The importance of interaction processes for the conceptual modelling of HC interaction is apparent: they are the points where human users provide information that is required for the work of the information system but which no other source can provide. Computations play a role in HC interaction modelling in so far as they provide support for the interaction, e.g., by computing suggestions for user input. HC interaction modelling is not, however, concerned with the algorithmic development of these computational processes but instead relies on their being provided by the process design component.

In our model, processes are modelled according to the following syntax:

```
⟨interaction⟩  ::= ⟨i_id⟩: IN (⟨cluster_list⟩), OUT (⟨cluster_list⟩),
                   ACT (⟨actors⟩)
⟨i_id⟩         ::= I#⟨number⟩
```

```
⟨computation⟩  ::= ⟨c_id⟩: IN (⟨cluster_list⟩), OUT (⟨cluster_list⟩),
                   ACT (⟨actors⟩)
⟨c_id⟩         ::= C#⟨number⟩
⟨cluster_list⟩ ::= CLUSTER (⟨conditions⟩): ⟨comp_list⟩
                   | CLUSTER (⟨conditions⟩): ⟨comp_list⟩, ⟨cluster_list⟩
⟨comp_list⟩    ::= COMP (⟨conditions⟩): ⟨did_list⟩
                   | COMP (⟨conditions⟩): ⟨did_list⟩, ⟨comp_list⟩
⟨did_list⟩     ::= ⟨d_id⟩
                   | ⟨d_id⟩, ⟨did_list⟩
⟨actors⟩       ::= ⟨a_list⟩ ? ⟨conditions⟩
                   | ⟨a_list⟩ ? ⟨conditions⟩, actors
⟨a_list⟩       ::= ⟨a_id⟩
                   | ⟨a_id⟩, ⟨a_list⟩
⟨a_id⟩         ::= A#⟨number⟩
```

Input data. Input data is data that is to be processed, accepted or stored. Potentially, each process can be connected with different input data. This is the case if

- interaction processes are to provide different data to the user depending on the interaction environment, i.e., to tailor the amount of information (*inclusion polymorphism*) based on user expertise (e.g., additional help information for novice users), interest (e.g., skipping sports information if the user is known to be not interested) and information already exchanged (avoiding unnecessary repetitions), or
- interaction or computational processes realise *ad-hoc polymorphism*, i.e., depending on the current state of interaction data of different (structurally incompatible) types is available and needs to be processed, displayed, etc.

For that reason, input data to processes is specified in a form that is similar to a disjunctive normal form:

```
IN (CLUSTER (⟨conditions⟩):
            COMP (⟨conditions⟩): ⟨data⟩,
            ...
    CLUSTER (⟨conditions⟩):
            COMP (⟨conditions⟩): ⟨data⟩,
            ...
    ...)
```

Clusters (CLUSTER) are alternatives and each cluster consists of several components (COMP). Components, however, are not compulsory but can be included and excluded, respectively, be conditions. The selection of a particular cluster is also controlled by conditions. Conditions can refer to all types of environmental information (social, technical, interaction and even stylistic aspects).

Clusters are usually used to realise ad-hoc polymorphisms (so that conditions referring to the interaction environment are often not required); whereas components are used to realise inclusion polymorphisms, i.e., to select information according to the interaction environment. In the following example the input part of an interaction process is described.

Example 4.2: Input data specification

```
I#1: IN (CLUSTER ():
                 COMP ():
                      D#10,
                      // refers to Customer_info data type
                 COMP (SOCIAL_MODEL:visits EQUAL 1):
                      D#11,
                      // refers to Help_for_Room_Info data type
         CLUSTER ():
                 COMP ():
                      D#20,
                      // refers to Room_Info data type
                 COMP (SOCIAL_MODEL:visits EQUAL 1):
                      D#21,
                      // refers to Help_for_Customer_Info data type
```

■

The expected output of this process could be some hotel reservation information (e.g., as specified in the earlier example on data types). The input could be already existing information on the customer or on the selected rooms—whatever is available at the current state of interaction. This information would by displayed to the user (that is why it is input information) as a support for the complete provision of the information required for a hotel reservation. In both cases, additional help information on how to fill in the respectively missing information is provided if the user works with the information system for the first time.

Output data. Output data is data which is the result of either interaction processes, i.e., is provided by the user, or of computational processes. Similarly

to input data, processes can be connected to several output data. This is the case if

- an interaction or a computational process is used to provide different output data depending on the overall interaction goal (ad-hoc polymorphism). These different output data can be associated as input data to different further processes so that various interaction flows can get realised.

The definition of output data is done according to the same scheme as for input data, i.e., in a disjunctive normal form like fashion.

Actors. Both interactions and computations are connected to actors specifying which users are allowed to perform a certain process. All users specified in the actor part of a process definition (ACT) will be enabled to start that particular process.

In simple cases, actors might correspond to user identities. Far more often, however, a more complex actor model is required that is able to capture substitutions (partially or completely substituting users, e.g., during their holidays), alternatives (e.g., collections of users with equal rights) and hierarchies.

The specification of actor models is not considered in HC interaction modelling but is part of the organisation structure of an application.

Connecting processes. Processes communicate via data so that in the modelling of HC interaction the relevant processes are connected via their input data and output data, respectively. This means that if output data of one process is input data for another process, both processes are connected. Figure 4.3 shows an example of three interaction processes which are eventually connected through the data entities (actually carrying identifiers instead of names) Customer_info and Room_info.

The model of process specification shown here is a simplified view on a general process design model. It is, however, sufficient for the context of HC interaction modelling. The general process design model is developed in cooperation with this work and proposes a number of additional concepts that are above all intended to further facilitate the usage and to increase the efficiency of the model [CLS99b, Cla00]. It is, in fact, similar to a Petri net model but introduces the following extensions (some of which have also been realised in various higher-order Petri net models):

Places are replaced by data elements which are described by types and manage a number of data values which are produced by their preceding transitions and consumed by their subsequent transitions.

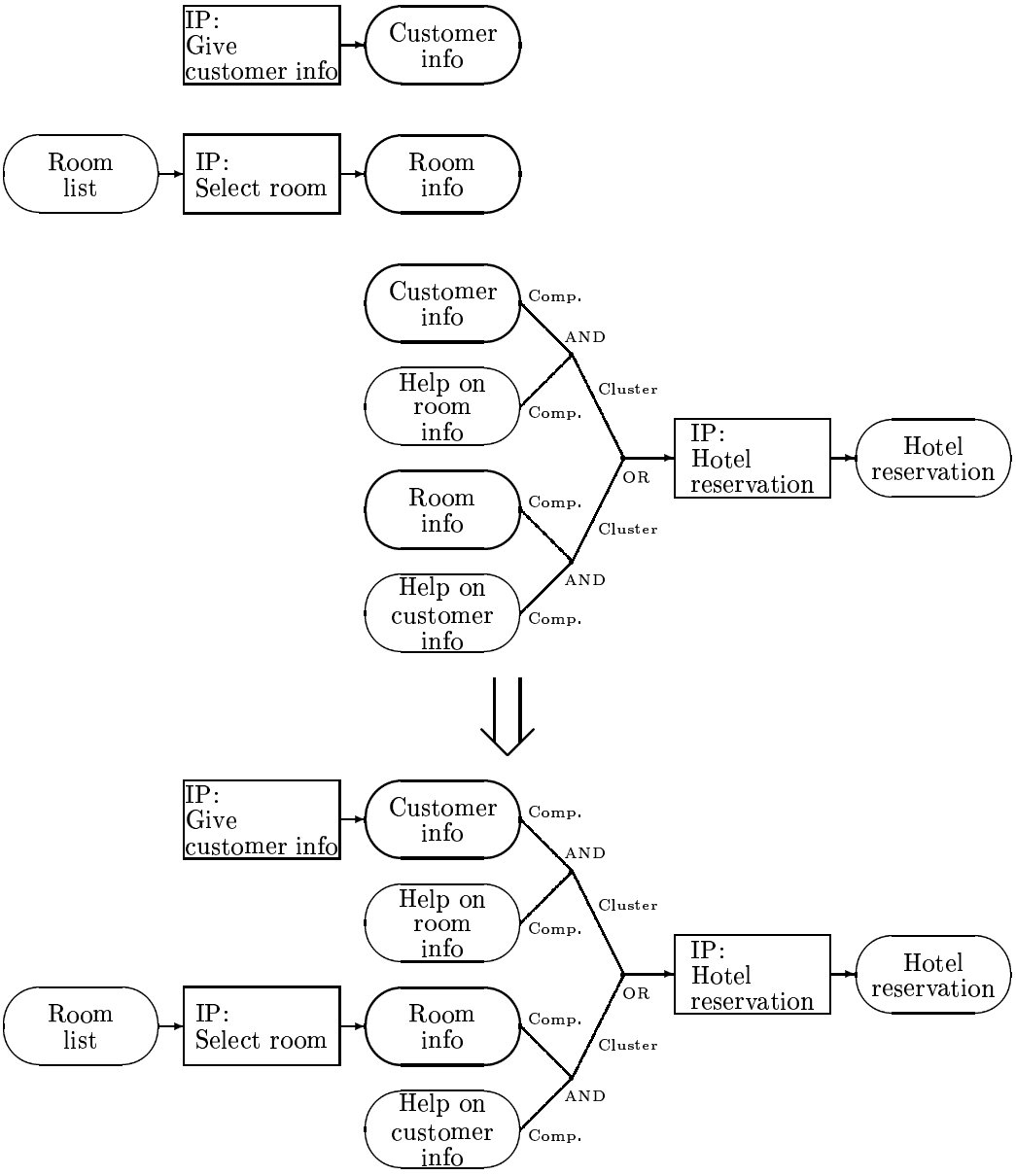


Figure 4.3: Connecting Processes

Invocation semantics describe the effects of the production and the consumption of data values. The following cases occur:

- A process f produces and writes data A. It can now either restart instantly to produce the next A (*eager call*) or it can wait for A to request a new value (*idle*).
- A requires a new value which is still to be produced. It can either wait for f to produce a new value (*passive wait*), it can start a parallel copy of f (*parallel lazy call*) or it can wait for f to terminate before starting it (*serial lazy call*).
- In A a new value is available for consumption. A can either wait for the value being fetched by a process f (*hold*), it can start a parallel copy of f with its current value as a parameter (*parallel eager call*) or it can wait for the termination of f before starting it (*serial eager call*).
- f enters the state of executability and needs to consume a value from data A. It can either wait for the arrival of a value in A (*active wait*) or it can ask for a new value to be created in A (*lazy call*).

Access semantics describe the effects of the read/write access of processes to data.

- In case of data consumption (read access) the value can be held after being read (*read-only*), it can be destroyed instantly (*'functional update'*) or it can be destroyed according to a customisable fashion, e.g., after a specified number of read accesses or after a certain period of time.
- When data values are produced (write access), existing values can be overwritten or they can be added and managed according to various mechanisms like FIFO, LIFO, priority queues, random choice, etc.

Association semantics have partially been realised in this HC interaction modelling language by means of the CLUSTER and COMP constructs. They describe what happens if more than two elements (i.e., more than one data entity and one process) are involved in a communication. E.g., in case of one data entity A being read by two processes f and g, the following associations could be specified:

- $\cup$ A is consumed by f as well as g (*copy*).
- $\cap$ A is consumed either by f or by g. Which one is chosen does not depend on the value of A but can be programmed to be *alternating*

choice, *random choice*, based on the system load (*load-balanced choice*), based on the states of f and g (e.g., *wait-duration choice*), etc.

$\times$ One pre-defined part of A is consumed by f, the other by g (*Cartesian product*)—this corresponds to clusters in the restricted view on the model used for HC interaction modelling.

$+$ A carries some information—typically the type of A—determining whether it is to be consumed by f or by g (*disjoint union*)—this corresponds to components in the restricted view on the model used for HC interaction modelling.

4.4 Characteristics and Relations

Data and processes can carry different semantical characterisations and can participate in relations [Lew99]. Characteristics and relations abstractly capture properties of data and of processes that provide an input to the evaluation phase, i.e., to the selection of appropriate interaction flows, contents and forms, and to the presentation phase, i.e., they influence the eventual visual/auditive appearance of HC interaction.

In the following, we present a motivation of such semantical data and process descriptions based on notions from linguistics and the concept of norms. Finally, we explore those concepts that are of relevance for the conceptual modelling of HC interaction. Table 4.1 summarises concepts for a semantic description of data, Table 4.2 for a semantic description of processes.

4.4.1 Motivation of Concepts

The abstract modelling of HC interaction requires above all to avoid the specification of concrete representations. This includes visual/auditive representations of data items, splitting or combining data items (depending, e.g., on the display size), the arrangement of data, etc. Instead, HC interaction modelling has to attempt to capture the *abstract properties of data and processes leading to such concrete representations*. In the ACE framework we use a number of semantical characteristics and relations for that purpose.

Abstraction in interaction. Abstract interaction concepts become apparent in all forms of communication: In common human intercourse, e.g., we usually device a picture of *what* to say before we use concrete words, phrase structures, etc., to address a particular interaction partner and use the media available. Interface design usually takes a similar approach.

The point is, however, that *(a)* we do not want to analyse the meaning of individual information items, their intended role in a particular interaction step, etc., in a separate and subsequent interface design activity and *(b)* we do not want to leave these often application-relevant decisions to the subjective opinion of the interface designer alone. We consider interaction design as an integral part of the overall application design; our goal, therefore, is to model the properties of data and processes that determine their eventual representation in conjunction with the abstract modelling of data and processes.

Natural language and norms. Natural language has always been appreciated as a source of intuitively understandable modelling concepts [BR95, Cou95]. In linguistics, notions of abstractively constituting the meaning of concepts and expressing relations have experienced an especially high level of scientific attention. While being aware of the lacking universality of linguistic notions [Who56], these notions can still be credited with a high degree of understandability and applicability.

Norms are a central notion of works on argumentation and also on deontic logics. They are used to describe the validity of claims, intentions, etc.

Besides existing work concerning the modelling of HC interaction (cf. Section 2.4), we will now gain motivation for semantical HC interaction concepts from the areas of linguistics and norms.

Natural Language

In human interaction, we do not use phrases, words or communication patterns because we have been explicitly and minutely instructed to do so. Rather we base our interaction contributions on some abstract notion of *what* to say and determine the way of how to express these things based on our knowledge of certain conventions. We then select presentations that we consider available, efficient and appropriate in a specific interaction situation. Linguistics has substantiated this rather intuitive concept of human communication with a number of theories that are of potential interest to this work.

Word semantics and semiotics. If we want to refer to objects, abstractions, states or actions in the real world, what do we do? We utter words, use gestures, write combinations of letters, etc. ¿From a semiotic point of view, these words, gestures and writings are representations (signifiers) of real world concepts (signifieds).

This distinction between the abstract concept and a concrete symbol used for its communication has originally been explored by de Saussure [Sau16]. Semiotics still is a much researched field of linguistics [Eco85]; the separation of concept and symbol can also be found in Bühler's Organon Model (cf. Figure 2.3, page 41).

Some approaches to interface design have already acknowledged the possibility of a semiotic approach [Nad88].

To make communication via representations (signifiers, symbols) possible we have to trust in conventions of sign usage that are established in every cultural context and are more or (often) less universal. When looking at the process of 'choosing' representations, it basically consists of three steps:

1. the establishment of the signified to be conveyed,
2. the selection of an appropriate convention of sign usage (considering the current interaction situation) and
3. the production of the signifier (materialisation).

Representations will eventually be *interpreted* by our hearers, readers or onlookers. To understand the communicated concepts they basically reproduce and invert the above process as is also shown in Figure 4.4. If the selected sign conventions do not match, misinterpretations and misunderstandings are the result.

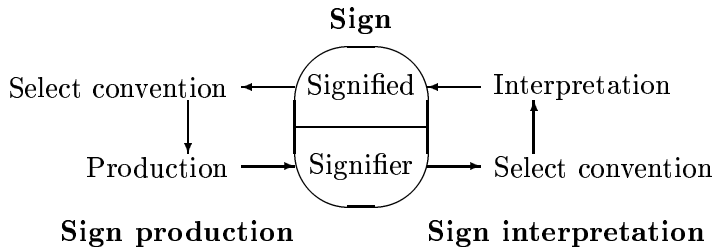


Figure 4.4: The Sign Process

Syntagmatic structures. Combinations of signs which have their own coherent meaning are called syntagmata. Syntagmata realise specialisations, i.e., particular concepts are restricted in their meaning. There are two ways of achieving such specialisations.

Firstly, concepts can be assigned to a *scale* which is often achieved by the use of adjectives or adverbial structures. When referring to the scale time we can, e.g., talk about an early flight and about a late flight or about a flight that is three times as expensive as another. A similar effect can sometimes be achieved by verb aspects (with Slavic languages being a good example).

Secondly, syntagmatic structures also play a role in establishing some kind of relations between text elements and often find expression in object structures

and clause relations. They can be used to specify a number of rhetorical relations (arguments, causalities, etc.) or more application-dependent relations like ownership (the account of the customer), place of departure (the flight from Alice Springs), etc.

Speech acts. The use of language usually aims at achieving a particular reaction or state of mind in the hearer. The hearer may be expected to simply accept the facts stated, to give information, to close a door, to feel frightened... The concept of using language to provoke actions is examined in speech act theory [Aus62, Sea69].

Searle [Sea69] distinguishes four types of speech acts:

- *the locutionary act* corresponding to the articulation of language elements in a certain grammatical order,
- *the propositional act* that establishes the content of an utterance by referring to objects of the real-world or the speaker's conceptional world and by assigning predicates (properties) to them,
- *the illocutionary act* that describes how the proposition is to be understood by determining the communicative function of the utterance and
- *the perlocutionary act* which captures the causal effects of the utterance, i.e., the reaction achieved in the hearer.

Consider, e.g., the following sentences:

Karla booked a hotel.
Did Karla book a hotel?
Please, book a hotel, Karla!

The *locution* involves the syntactical formation (as shown above) and the utterance of these sentences as, e.g., *[k[a']rl[a'] haz bukt A hOtel]*[1]. In all sentences, the *proposition* then consists of the reference 'Karla' and the predication 'book a hotel'. The illocution, however, differs: It is a simple statement in the first sentence, a question in the second and a command in the third. Possible perlocutions are the answers 'So did I.' or 'She did indeed.' in the first case, the answer 'Yes, I'm sure she did.' in the second case or Karla's rushing to the telephone in the third case.

The most prominent and probably most discussed feature of speech act theory are the illocutionary acts that describe the way language is used to invoke the desired reaction from the hearer. Searle identified five illocutionary forces:

[1] According to the Pronunciation Guide of the WWWebster Dictionary [WWW].

- *assertives* used by the speaker to identify him/herself with the truth content of the uttered fact,
- *directives* used to initiate a particular action from the hearer,
- *commissives* used by the speaker to commit him/herself to some future activity,
- *declarations* used to establish a fact as true by uttering it and
- *expressives* used to state a particular mental attitude.

The same perlocution can be achieved by means of different illocutions. In order to make somebody close a door I can, e.g., use an assertion (*I am cold.*) or a directive (*Close the door!*).

This rather rough classification has occasionally been redefined in an attempt to achieve better formalisation. The extensive classification of English verbs by Ballmer and Brennenstuhl [BB81] is such an example which has resulted in a detailed categorisation of verbs according to the activities they describe and the *purpose* of these activities.

Text organisation. Texts are characterised by the existence of an internal structure, i.e., relations between individual text parts. It is textual relations that make the interpretation of a text in its entirety possible.

An approach to the description of the internal structure of (written) texts is *Rhetorical Structure Theory* [MT87]. It provides a systematic definition of relations that can hold between parts of a text. Some examples of such relations are elaboration, circumstance, solutionhood, purpose, condition, motivation, evidence, etc. The relations are application-independent throughout. They do not only apply in texts but also in less complex interaction contributions.

Conversational obligations and dialog acts. Just as important as the relations within texts or individual interaction contributions is the interplay between different interaction contributions. Conversational analysis [Lev83] provides a general background to the topic of information exchange. It empirically examines mechanisms of turn taking, adjacency pairs (question–answer, offer–acceptance, etc.) and the overall structural organisation of conversation (e.g., greeting–main part–closing). It is above all the concept of adjacency pairs that allows to describe expectations resulting from the utterance of particular types of dialog contributions (e.g., suggesting a list of values–selecting a value).

A more focussed approach is that of Alexandersson et al. [ABF+97] which tries to characterise interaction contributions according to their role in information exchange. Their classification of dialog acts includes conventional acts (greetings,

thanks), informing acts, requests, suggestions and feedback. With regard to information exchange from the system's point of view, the classes of request and suggestion are of special interest. These put obligations on the dialog partner in so far as they restrict the set of acceptable answers. As conversational obligations in general, they can also introduce integrity constraints as, e.g., the request to pick an answer from a given set of possibilities.

Norms

Theories on argumentation and on deontic logics, too, have the concept of norms at their core [Fac92, Wri77]. By establishing norms claims, intentions and propositions can be judged upon in terms of their validity.

When modelling systems, we basically describe norms for the system behaviour, thus in turn establishing norms for the behaviour of human users, too (since human users can only do what the system allows them to do).

In heterogeneous environments then, the possibility of a heterogeneity of norms must also be given. This has already been realised in the incorporation of conditions when connecting data as input data and output data to processes. Such conditions also need to be allowed for all further semantic properties of data and processes so that in dependence on the current interaction situation only certain features are allowed while others are forbidden.

4.4.2 Data Characteristics

Data characteristics extend the modelling of interaction processes by describing at a more detailed level the meaning of participating data. They also weigh data with regard to different application-dependent and application-independent criteria.

Technically, we distinguish between direct and indirect data characteristics. Both are described by an identifier, the data they refer to, a type, a concrete value, the interaction process they relate to and conditions that control their applicability. Indirect characteristics are moreover associated with visibility and priority attributes.

```
⟨dir_char⟩   ::= ⟨ch_id⟩: ⟨d_id⟩, ⟨dch_type⟩, ⟨quantity⟩,
                 ⟨context⟩, ⟨conditions⟩
⟨indir_char⟩ ::= ⟨ch_id⟩: ⟨d_id⟩, ⟨ich_type⟩, ⟨quantity⟩,
                 ⟨context⟩, ⟨conditions⟩,
                 ⟨priority⟩, ⟨visibility⟩
⟨ch_id⟩      ::= CH#⟨number⟩
```

```
⟨dch_type⟩    ::= MANIP
                | SELECT
                | RELEV
                | EMPH
⟨ich_type⟩    ::= ALERT
                | APPL:⟨word⟩
⟨quantity⟩    ::= ⟨ch_equation⟩
                | ⟨ch_range⟩
                | ⟨ch_comparison⟩
```

Types. We distinguish qualitative, quantitative, usage and functional characteristics.

- *Qualitative characteristics:*
  Motivation: semiotics;
  Data can be assigned to specific application-dependent concept categories such as, e.g., time, place, customer, etc. (APPL:⟨word⟩). Principally, the types of data can also be considered as such concept categories. Qualitative characteristics, however, serve as an additional categorisation.

- *Quantitative characteristics:*
  Motivation: syntagmata;
  Data can also be *scaled*, i.e., their correspondence to several application-independent and application-dependent criteria can be measured.

  Application-independent criteria are relevance and emphasis. Relevance determines how urgently data has to be made available, e.g., made visible or audible. If data is made available, emphasis then controls how prominently that is done.

  Application-dependent criteria are introduced according to the application domain in question. Possible criteria are age, duration, expensiveness, etc.

  Quantitative characteristics can be measured (⟨quantity⟩) absolutely or in dependence on each other. On the one hand, one can, e.g., specify an emphasis of 95 (on a scale of 0 to 100). On the other hand, one can specify that some data A has a higher emphasis than or an emphasis three times as high as that of data B, that a flight is more expensive or a book is older than another.

- *Usage characteristics:*
  Motivation: speech acts;
  Data has to fulfill certain functions which are mainly information (in), which

corresponds to the role of input data, and reaction (out), which corresponds to the role of output data.

Some input data, however, is not meant as mere information but fulfills a specific role, namely the invocation of some activity on the user's side. This is achieved by assigning the characteristic alert.

- *Functional characteristics:* As part of the interaction semantics, data retrieved from the underlying database can be customised to the user to be addressed and/or to the current interaction state. As a consequence, a large amount of in fact interaction-related content adaptions can be kept out of the general process modelling and be considered in the context of HC interaction modelling instead.

  Firstly, by means of the manipulation characteristic, data can be 're-calculated', e.g., students can be automatically offered discounts, system administrators can be given starting times minus one hour to make sure they are at least somewhat in time, etc. Secondly, by means of the selection characteristic, data can be selected, e.g., only high-priced products are displayed for wealthy or price-insensitive customers.

Conditions. The assignment of characteristics to data can be controlled by conditions that take into consideration the properties of users and the goal and history of interaction. Thus it is possible to customise the emphasis to be put on data, the values to be used for data instantiation, values used as suggestions for user input, etc.

It has to be noted that above all the customisation of characteristics and values (but also the customisation of scenarios, etc.) can be a delicate topic—especially if they are used to the disadvantage of customers. The use of that notorious small print in contracts or online booksellers that could add 20% to prices for customers that are known not to be very sensitive about prices are only some unpleasant examples.

However, customisation mostly *is* used to the customer's advantage, and subjective misuse of this feature can unfortunately not be ruled out by a design methodology. Hence, just as in human intercourse where misinformation, etc., is also practised, credibility of information services is an important aspect. Techniques can and should be installed that allow an assessment of credibility, e.g., certification by independent organisations, exchange of personal experiences, etc.

Priority and visibility. For some characteristics one can in addition specify the two properties of priority and visibility. These properties are applicable for qualitative, quantitative application-dependent and usage characteristics (indirect characteristics).

Basically, they correspond to the characteristics relevance and emphasis: The priority property determines how important it is to make a particular characteristic apparent. If a characteristic is to be made apparent, the visibility property decides how prominently this characteristic is then displayed. It is obvious now that these properties are not applicable for functional characteristics and that there would be no point in applying them to the relevance and emphasis characteristics themselves (direct characteristics).

Both priority and visibility can again be controlled by conditions.

4.4.3 Data Relations

Data do not only have there own particular meaning and scaling but they also take part in relations between each other. These relations are described by an identifier, by the data functioning as source and as target of the relation, by a type, a context, conditions, priority and visibility. Technically, we distinguish flat relations and hierarchical relations. Hierarchical relations can be applied recursively.

```
⟨flat_rel⟩   ::= ⟨rel_id⟩: SOURCE ⟨d_id⟩, TARGET ⟨d_id⟩,
                 ⟨flat_type⟩, ⟨context⟩, ⟨conditions⟩,
                 ⟨priority⟩, ⟨visibility⟩
⟨hier_rel⟩   ::= ⟨hrel_id⟩: SOURCE ⟨part⟩, TARGET ⟨part⟩,
                 ⟨hier_type⟩, ⟨context⟩, ⟨conditions⟩,
                 ⟨priority⟩, ⟨visibility⟩
⟨rel_id⟩     ::= R#⟨number⟩
⟨hrel_id⟩    ::= HR#⟨number⟩
⟨part⟩       ::= ⟨d_id⟩
                 | ⟨hrel_id⟩
⟨flat_type⟩  ::= ELAB
                 | SUMM
                 | CONTEXT
                 | HELP
                 | SUGG
                 | CHOICE
                 | DEFAULT
                 | APPL:⟨word ⟩
⟨hier_type⟩  ::= GROUP
                 | SEQU
```

Types. For data relations we distinguish rhetorical relations and organisational relations.

- *Rhetorical relations:* Connections between data that enable an orientation along the logical structure of interaction are called rhetorical relations.

  - *Narrative relations* (*motivation:* text organisation) describe the inner structure of data that the system provides to the user, i.e., of the input data in interaction processes. Typical examples are elaboration, context, help or summary.

    Narrative relations are not restricted to single interaction processes but can also be used to initiate further contributions from the system. Help data or summaries may, e.g., not be displayed in the first place but may be provided only on the user's request or after an analysis of the user's behaviour. In any case, the logical connection between 'core' data and narratively related data must be appropriately conveyed.

  - *Conversational relations* (*motivation:* conversational obligations, dialog acts) are established between input data and output data of interaction processes, i.e., between data the user is provided with and data the user is expected to give.

    The main use of conversational relations is the direct support of the user's input. Typical examples are suggestion, default, explanation or choice. In the cases of suggestion and default the user is suggested a potential input value or a set of potential input values and can accept that suggestion, choose a suggested value and insert a new value, respectively. In the case of the choice relation, a discourse obligation (integrity constraint) is expressed as the user is confined to selecting a value from a given set.

- *Organisational relations:*
  Motivation: syntagmata;
  The arrangement of data is determined by organisational relations which can be either application-dependent or application-independent.

  Application-dependent are relations like ownership or time of departure (for flights). They exactly specify the type of relation between the data involved.

  More universal and less concerned with semantic subtleties of the respective application domain are application-independent relations as, e.g., groupings and sequences. Instead of exactly stating the kind of relationship, they simply specify that several data 'belong' together or that data is to be provided/elicited in a particular order. Organisational application-independent relations are also called *hierarchical relations* as they can be specified in a recursive fashion.

Conditions, priority and visibility. Similarly to data characteristics, relations can be controlled by conditions so that different user types and different interaction states can be given credit to.

Moreover, the properties of priority and visibility are applicable to all relations. They decide whether a relation is to be made apparent and, if so, how strongly apparent it is to be made. Again, conditions can be applied to these two properties.

Type	Subtypes	Examples
Data characteristics	Qualitative	time, place, customer, ... (application-dependent)
	Quantitative	relevance, emphasis (application-independent) duration, expensiveness, age, ... (application-dependent)
	Usage	alert (application-independent)
	Functional	manipulation, selection (application-independent)
Data relations	Rhetorical: Narrative	elaboration, summary, context, explanation, help (application-independent)
	Conversational	suggestion, choice, default (application-independent)
	Organisational	grouping, sequence (application-independent) time of departure, ownership, ... (application-dependent)

Table 4.1: Semantics of Data Entities

4.4.4 Process Characteristics

Just as data, processes can carry a number of characteristics that further describe their meaning and usage.

⟨p_char⟩ ::= ⟨p_ch_id⟩, ⟨p_id⟩, ⟨p_ch_type⟩, ⟨quantity⟩,
⟨conditions⟩

```
⟨p_ch_id⟩    ::= R_CH#⟨number⟩
⟨p_id⟩       ::= ⟨i_id⟩
                 | ⟨c_id⟩
⟨p_ch_type⟩  ::= RELEV
                 | PRIOR
                 | APPL:⟨word⟩
```

Types. For processes, we distinguish qualitative and quantitative characteristics.

- *Qualitative characteristics:*
  Motivation: speech acts, verb classification;
  Processes can be characterised by their purpose—a criterion which is always application-dependent. Typical examples are booking, browsing, etc.

- *Quantitative characteristics:*
  Motivation: syntagmata;
  Furthermore, processes can get assigned the criteria of relevance and priority which on the one hand describe how important it is to execute a process and on the other hand how urgent the execution is.

  If *relevance* is specified in dependence on the state of other processes, causal dependences can be described.

Conditions. Qualitative as well as quantitative process characteristics can again be specified in dependence on the current interaction situation as has already been briefly mentioned. Thus it is possible to have different interaction flows created depending on the intentions of the user.

4.4.5 Process Relations

Process relations describe different processes in dependence on each other. Here, we are concerned with *organisational relations* only, i.e., group and sequence. These will help to ensure coherence of interaction since normally processes could, e.g., be started whenever there input data is available. Such a strategy would often lead to illogical jumps in the overall flow of interaction so that some kind of organisation is strongly required.

In the case of groupings, processes are organised into groups within a (mostly semantical) context, e.g., all processes that relate to the search for a hotel on the one hand and all processes that concern transportation on the other hand. In the case of sequences, an order between processes is specified such as the extraction of customer information before the extraction of further search criteria.

Process relations are specified according to the following syntax:

```
⟨p_rel⟩     ::= ⟨p_rel_id⟩: SOURCE ⟨r_part⟩, TARGET ⟨r_part⟩,
                ⟨hier_type⟩, ⟨conditions⟩
⟨p_rel_id⟩  ::= P_R#⟨number⟩
⟨r_part⟩    ::= ⟨p_id⟩
                | ⟨p_rel_id⟩
```

Conditions. As usual, process relations can be specified in dependence on the interaction situation. The flow of interaction can thus be further customised to the user's requirements.

Type	Subtypes	Examples
Process characteristics	Qualitative	booking, browsing, ... (application-dependent)
	Quantitative	relevance, priority (application-independent)
Process relations	Organisational	group, sequence (application-independent)

Table 4.2: Semantics of Processes

4.5 Modelling Constructs and the Components of Human-Computer Interaction

In the previous section we have presented a number of constructs for the abstract and complete modelling of HC interaction, i.e., of the interaction flow, the interaction content and the interaction form. Taking a broad perspective, there are correspondences between

- interaction flow and constructs for the modelling of processes,
- interaction content and constructs for the modelling of data as well as
- interaction form and constructs for the modelling of characteristics and relations.

Looking more closely, however, these correspondences are not one-to-one. In the following, we point out how the proposed modelling constructs relate to the individual components of interaction.

Interaction flow is modelled by

- processes (especially the connections between processes and data),
- characteristics and relations of processes that control the actual selection and sequencing of processes.

Interaction content is modelled by

- data,
- qualitative, quantitative and functional data characteristics and
- narrative data relations that all influence the selection of data and (in the case of functional characteristics) also their instantiation.

Interaction form is modelled by

- qualitative, quantitative and usage data characteristics as well as
- all data relations, which influence the eventual appearance of interaction, i.e., visual/auditive properties of interface constructs and their arrangement.

The result of HC interaction modelling is a bipartite hypergraph consisting of data elements and process elements, associated properties of data and processes and complex (branched) connections between them. Such an HC interaction model can be the starting point for an automatic generation of customised interaction. During this generation process, *interaction objects* will be derived and physical presentations for these interaction objects will be created that are tailored to the current interaction situation. An ordered processing of a number of interaction objects represents an interaction instance. The process of deriving interaction objects and their physical presentations is covered in Chapter 6.

4.6 Modelling the Environment

An approach that is aimed at a customisation of HC interaction requires knowledge on the interaction environment. Although for our work, we assume the existence of environment models, we want to provide a short discussion of important aspects of environment modelling. Knowledge that can be of relevance to interactive tasks relates to the following factors:

- social factors (characteristics of the users and their cultural and domain-specific backgrounds),
- technical factors (media and interaction channels available),
- stylistic factors (guidelines for interaction appearance) and
- interaction factors (interaction goals, history, etc.).

Which factors are actually collected depends on three issues: *(a)* which information on users can be extracted with reasonable efforts and without decreasing user acceptance (i.e., annoying the user), *(b)* which information is of relevance to the particular application and *(c)* which information is (technically) usable for the adaption of the system.

For those factors that were chosen to be used for a customisation of HC interaction, it is then of importance to

- *obtain* information on these factors,
- to *represent* and *maintain* that information and
- to *utilise* it for a customisation.

It is the social factors of interaction environments that have so far received the most intense attention in the context of adaptive systems. Especially user modelling [Kob93, SKM93], i.e., the extraction and exploitation of user descriptions, is a field where much theoretical as well as practical research has been conducted. The mechanisms available for other factors of interaction environments are, however, similar.

In the following sections we will discuss how information that is of potential interest to a customisation of HC interaction can be gathered, which models exist for its representation and maintenance and which approaches to its utilisation exist. A more general discussion of the influence of environmental factors on the materialisation of HC interaction is then presented in Chapter 6 which deals with the automatic generation of HC interaction.

4.6.1 Gathering Information on the Interaction Environment

In literature on user modelling we find the suggestion that user modelling involves the description of *typical users*, of *user groups* and of *individual users* [Kob93, DMKS93]. For all three categories their

- needs and preferences

have to be modelled. In addition, individual users can be further described by their

- characteristics,
- abilities,
- interests,
- behaviour,
- personal knowledge and
- experience [DMKS93].

For other factors of interaction environments such classifications of criteria do not seem to exist but Section 1.3 of this work has already provided an overview of further relevant features such as

- conventions originating from the cultural or geographical background of the user,
- the interface system to be addressed, its capabilities (e.g., plug-ins and additional devices) and the capacity and reliability of the interaction channel (technical factors),
- stylistic guidelines determining the eventual appearance of interfaces (stylistic factors) and
- interaction factors such as the state and the goal of interaction.

Especially in the cases of social, technical and interaction factors, ethical questions are raised concerning the privacy of users which should always be respected. These questions have to be considered too when it is decided which information to obtain at all.

Information on the interaction environment can be extracted either directly or indirectly.

Direct gathering. Direct information is gained through explicit user statements (in questionnaires, system options, etc.), through a checking of the system state (environment variables) or through an observation of the user behaviour (error rate, typing speed, etc.).

Besides explicit user interrogation it is, e.g., possible to use JavaScript to detect the Web browser software consumers are using [WL98] so that special capabilities of each browser can be exploited. Similarly, the operating system

used and a number of software available can be determined. An extreme case is that of a company which recently admitted that a music software distributed by them surreptitiously reported back on users' identities, listening habits and the number of songs stored on the users' hard drive [Smi99]—another incident of ignoring the privacy of users.

Direct information on the interaction environment is usually very reliable but often not sufficient to perform an effective customisation.

Indirect gathering. That is why indirect information is tried to be gained by analysing and reasoning on direct information; it must be considered less reliable than direct information and should be handled appropriately.

Some examples of indirectly inferred information are the user's location and occupation based on an analysis of the domain name of the server used (.au, .de, .uk; .ac, .edu, .com, etc.). Information on the user's intentions and purpose can be gathered by an extraction and analysis of behaviour patterns [SFW99, ZXH98].

4.6.2 Models for the Representation and Maintenance of Environmental Information

In order to represent information and the interaction situation encountered simple models like Cookies in the case of Web browsers can be employed. More sophisticated models originate again from the domain of user modelling where approaches are classified into quantitative and qualitative models [DMKS93].

Representing information. *Quantitative models* contain explicit information on individual users and/or user groups. E.g., in scalar presentations, flags can be used to mark whether users have particular characteristics or not [Kay93]. Hierarchic presentations allow to inherit and specialise information within the model and to define dependences [Kay93]. Stereotypes are one of the earliest presentations of user models [Ric79]. They are partial descriptions of frequently occurring situations that capture significant aspects but can ignore less important ones. In the modelling language proposed in Section 4 we assume the existence of such a stereotypes-based environment model.

Qualitative models are based on so-called overlay or differential techniques. They represent user characteristics, especially user knowledge, as a subset of or as a deviation from a general model of characteristics and of the system knowledge, respectively [BS97].

Maintaining information. Especially when indirectly gathered information is incorporated into an environment model, inconsistencies can arise, i.e., information found at a later point of time may contradict information that had

been included before. To make matters worse, other inferences may have been made based on the information that is now found to be incorrect so that a complete 'pruning' of all the information contained in the model has to be done.

Theoretical background for the handling of inconsistencies is provided by non-monotonic reasoning which uses the concept of assumptions in the case of lacking evidences [Łuk90].

4.6.3 Utilising Environmental Information

The aspect of exploiting information that has been gathered about the current interaction situation surely is one of the most interesting. In the following, we will present examples of approaches that have looked at the different factors of interaction environments. A more focused discussion in relation to a concretisation of the abstract modelling constructs proposed before is provided in Section 6.

Social factors. Brusilovsky has pointed out that an adaption to different users can occur along the two dimensions of *presentation* and *navigational support* [Bru96]. Although he considers adaptive hypermedia systems in particular, this classification is valid for other information system applications as well.

Adaptive presentation involves the customisation of content and the combination of multi-media elements in a fine-tuned way. *Content* can be adapted to the interests and goals of users as, e.g., discussed by Eklund and Zeilinger [EZ96] or as implemented in the AVANTI information system [FKN97a, FKN97b] where elderly users tend to be presented with information on cultural events rather than sports and modern entertainment. The AVANTI system also exploits user information to adapt the *prominence and level of detail* of the information provided. Furthermore, user characteristics can be used to adapt the *functionality* offered by user interfaces to match the expertise of the respective user [Bru95, BS97]. André et al. attempt a combination of different multi-media objects that suits the preferences of individual users of information systems [AFG+93].

Adaptive navigational support—especially in hypermedia systems—can be realised by the provision of explicit guidance, by the sorting and/or hiding of links, by an appropriate annotation of links, etc. This still is a relatively new area of research [Bru96, FKN97b].

Technical aspects. The need to take different technical aspects of interaction environments into consideration has only recently been recognised which is mainly due to the increased significance of networked environments. Whereas formerly technical aspects were important only in so far as, e.g., handicapped users had to be catered to (Braille displays for blind users, etc. [FKN97b]), the challenge is now to supply information via a vast range of varying technical equip-

ments and to cater to access methods as diverse as graphical interfaces, cellular phones, natural language interfaces, etc.

To capture the capabilities and constraints of target platforms it is important to describe *(a)* data that can be represented on the interaction medium, *(b)* functionality provided by the interaction medium, *(c)* the size or memory capacity of the interaction medium and *(d)* the capacity of the interaction channel. This information can be stored in a set of vectors [LS00].

With regard to exploiting the capabilities of the addressed interaction medium, Lansdale and Ormerod suggest the distinction of three different aspects of display development: the layout, the presentation and the coding [LO94]. Although this distinction is clearly motivated by the design of graphical interfaces, it can be adopted for HC interaction design in general and can support the separation between the abstract and the concrete levels of HC interaction with regard to the interaction form.

- *Layout* is concerned with the arrangement of information on the screen and—if we extend this notion from graphical interfaces to interfaces in general—with the determination of sentence structures, the organisation of 3D spaces, etc. Besides user characteristics like cognitive abilities, etc., layout is dependent on technical characteristics, e.g., the screen size.

- *Representation* deals with the selection of graphical constructs (e.g., tabular vs. diagrammatic representations, etc.), individual labels and terms, even language, etc. It is influenced by the set of constructs available on the targeted interface (i.e., the interface device language), by the ability of the interface to render graphics or sounds [AFG+93], by the capacity of the interaction channel, etc.

- *Coding* is the process of mapping information semantics to certain physical dimensions, e.g., importance to colour, profits to the y axis of a diagram, etc. Again, coding depends on the availability of physical dimensions on the targeted interface system. E.g., on a monochrome screen it would be more appropriate to map importance to font size instead of colour.

Stylistic aspects. Flow, content and above all form of interaction are also influenced by general design decisions. Such design decisions are motivated either by the consideration of ergonomic issues or by particular design philosophies mirroring corporate identities, etc.

Much work has been conducted on stylistic issues in so far that guidelines for effective and efficient interaction have been established. What is mainly not considered is the support of guideline modifications. Technologies like Cascading

StyleSheets [CSS] aim at providing a means for an easy exchange of stylistic properties; they still move, however, on the level of mere interaction form.

- *Ergonomics and cognitive psychology* stress the importance of aspects like short memory capacity, visual capacity, etc., that have to be considered for efficient communication [Urb91]. The interpretation of interfaces is mainly influenced by what users expect based on former experiences; similarly, the familiarisation with new interfaces is supported by the use of known metaphors and analogies.

- Insights from the design of *print media* or even from design in general also have to be observed [MS95]. These concern the readability of fonts, suitable combinations of colours, the power of certain physical dimensions to visualise contrasts, rules for the spatial arrangement of information, etc.

- Another source of stylistic aspects are certain design philosophies. They might intend to convey a *corporate identity*, i.e., to produce a certain image or to improve the recognisability of a trade mark or a company. Design philosophies can also be motivated by *aesthetical concerns*. Especially Web sites (e.g., Atlas [Atl]) occasionally go to great pains in order to transport exactly the design they want and to create identical appearances on different Web browsers.

The observation of ergonomic, cognitive and design guidelines aims at ensuring *usability* [Nie93], i.e., the fitness of computer applications for use by human users. Design philosophies and user preferences or requirements may sometimes contradict usability. In that case either of them has to be prioritised based on the current context.

Interaction aspects. Interaction aspects become manifest in different interaction goals (intentions) and histories.

- *Interaction goals* are mostly assumptions of the system that concern the tasks the user intends to fulfill [YM94, Pol92]. Naturally, interaction content and also interaction flow and form are influenced by such information. E.g., the fisheye principle [Fur86] can be used to distort an information display in such a way that information which is considered particularly important at the current point of interaction is shifted towards the center of attention. Later applications have used this mechanism to also restrict the amount of information presented [Mit90]. Moreover, assumed goals and tasks can be exploited in order to generate suggestions to the user as a form of guidance [Hop93].

- Knowledge on the *interaction history* is above all used to control the information to be presented. Background information and context information do not always have to be part of the information provided to the user. They should, however, still be retrievable on demand and should also be occasionally refreshed on the system's initiative.

4.6.4 Requirements for a Customisation of Human-Computer Interaction

In order to enable a customisation of HC interaction, both the modelling and the automatic generation of HC interaction need to meet certain requirements:

- When *modelling* HC interaction, it should be avoided to fix on certain interaction flows, contents or forms whenever possible. Instead, *abstract characteristics and relations* can be used to describe the actual interaction semantics and to guide the eventual selection of relevant aspects. Another possibility is the use of *conditions* to explicitly control the appropriateness of different flows, contents and forms.

  The process of modelling customisable HC interaction will be covered in Section 5.

- The *generation* of HC interaction needs to fulfill two tasks: Firstly, in a concrete interaction situation, it has to derive an *interaction object* (or a set of interaction objects) that fits the concrete situation, i.e.,

  - that comprises contents according to the user's interests and requirements, to the presentational capabilities of interaction media and channels and to the requirements of the interaction goal and history,
  - that contains characteristics and relations between content entities that correspond to the user's preferences (mostly in terms of an eventual presentation) and that can be presented on the interaction medium available and
  - that offers transitions to further interaction objects that the user would be interested in, require or prefer or that the actual goal of interaction would require.

  Secondly, the generation of HC interaction involves an appropriate *physical presentation* of the derived interaction object(s)—appropriate in terms of what is technically possible and what is wished for by stylistic guidelines and/or user requirements and preferences.

  Issues of the automatic generation of customised HC interaction will be discussed in Chapter 6.

Chapter 5

The Process of Developing Human-Computer Interaction

In Sections 3 and 4 we have introduced the ACE framework which provides means for the design of flexible, i.e., customisable and extensible, HC interaction in heterogeneous and dynamic environments. The ACE framework accomplishes two things: Firstly, it establishes a meta model for the conceptual modelling of HC interaction. The resulting abstract HC interaction model is to a large extent independent from specific interaction situations, especially from specific technical equipments. Secondly, the ACE framework structures the components that are involved in the materialisation of HC interaction, i.e., the transformation of an abstract HC interaction model into a physical presentation. This materialisation process is influenced by a number of environmental factors which are kept in various environment models.

The following chapter describes *how* the individual ingredients, i.e., the abstract HC interaction model and the environment models, are developed. It gives account of the activities as well as the actors involved in the process of HC interaction development.

Section 5.1 describes the abstraction layer Co-Design model, Sections 5.2 and 5.3 describe the design of HC interaction within this model. They suggest a number of design steps and show dependences between HC interaction development and other design aspects like view design and behaviour design. In Section 5.4 we then set forth how design tasks are distributed between different design experts.

5.1 The Abstraction Layer Co-Design Model

A design is in principle the description of the solution to a problem. Typical problem areas that are attempted to be countered with the help of computer application systems are inefficiencies or unproductiveness in workflows, the ex-

istence (or definition) of information needs, entertainment needs, etc. A design then describes a possible solution to the stated problem in the form of a computer application system.

Arriving at a design, i.e., a solution, is usually not trivial, especially if the problem is complex—which many problems tend to be.

To facilitate the design process, i.e., the process of finding and formulating a solution, design models are used that guide the design process according to a particular strategy. In the following, we present the abstraction layer Co-Design (alCoD) model which has been developed for an integrated design of different design aspects like data structure, views, behaviour and HC interaction [Tha97, CT97]. In Sections 5.2 and 5.3 we then describe the design of HC interaction within the alCoD model. We explore the HC interaction aspect of the abstraction layer Co-Design model by assigning design tasks to design layers and we highlight mutual dependences between HC interaction design on the one hand and view design and behaviour design on the other hand.

The abstraction layer Co-Design model. The abstraction layer Co-Design (alCoD) model has been developed for the design of high-level information systems. These applications are characterised by the need to manage a relatively extensive database, by a high amount of HC interaction and by a medium amount of functionality offered both for data processing and navigation. Typical examples of such high-level information systems are city and tourist information systems, booking and reservation systems or information brokering systems (news, share prices, etc.)

In Section 1.6.1 we have identified the following design aspects that need to be considered in the design of such high-level information systems:

- data structure (i.e., database schema),
- view structure (i.e., definition of user views),
- behaviour (i.e., the entire functionality) and
- HC interaction (i.e., communication with human users).

It has been pointed out that the use of different models and languages for the development of each of these aspects implies a high risk of producing incomplete, inconsistent and non-optimal designs and implementations of application systems [Tha97, CT97]. The Co-Design methodology has been proposed for an integrated design and development of different system aspects [Tha97, CT97, CLT97, FST98]. Its main contribution is the support of design methods for a parallel and fine-tuned development of data structure, views, behaviour and HC interaction. An integrated system design promises to significantly improve the

productivity of the design process as well as the maintainability of the application system.

The design layers. The abstraction layer Co-Design (alCoD) model [Tha97] is shown in Figure 5.1. The layers of this model mirror different viewpoints and levels of detail. The alCoD model does not, however, define a particular sequence of these levels and can in fact be used to produce different design strategies.

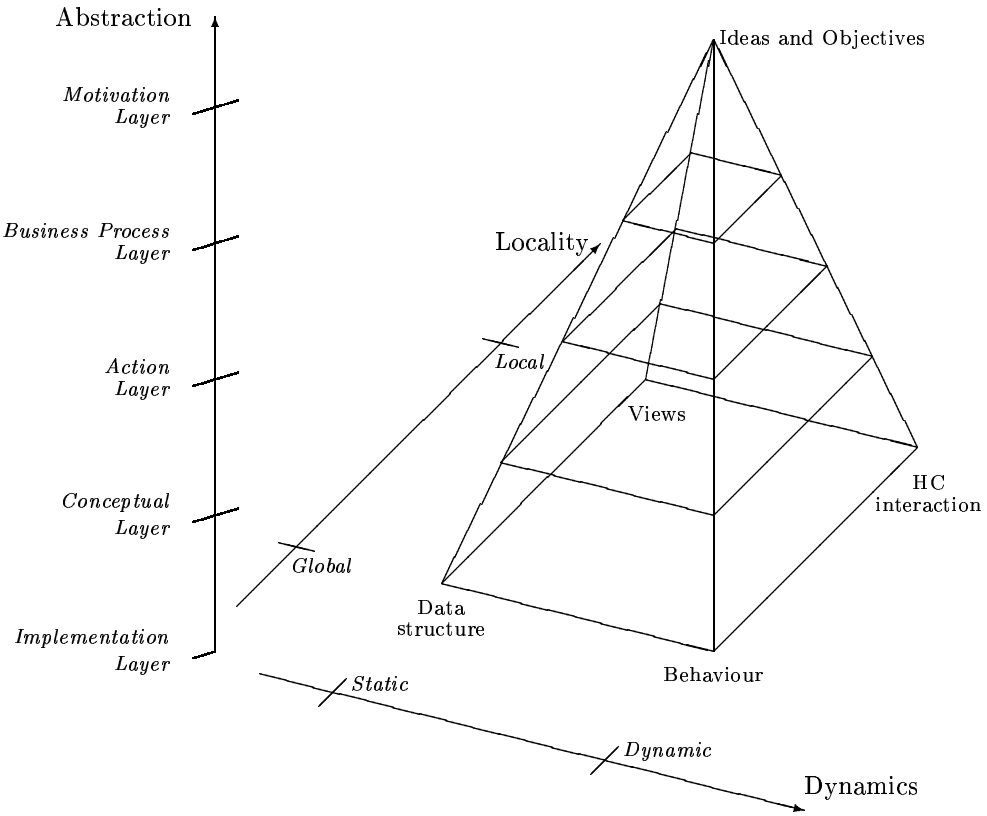


Figure 5.1: The Abstraction Layer Co-Design Model

On each layer the different design aspects of data structure, views, behaviour and HC interaction are developed in parallel and under consideration of their mutual dependences. The distinctions between the individual aspects become more apparent the more concrete the design becomes. The design layers are described in the following. HC interaction modelling within the alCoD model will be discussed in more detail in Section 5.3.

Motivation layer: This most abstract layer describes the ideas and objectives behind the information system. It is, so to speak, the formulation of the

problem to be solved by means of the information system. This involves an analysis of the application domain and its current structure as well as the delimitation of the solution. Business strategies to be supported are manifested and the required resources and the feasibility of the project are determined.

The motivation layer also includes a description of the intended effects of the solution. The specification of intended effects—as, e.g., the increase of sales or customer numbers by x%, the increase of the number of positive customer replies, etc.—are necessary to evaluate the eventual solution.

Business process layer: The business process layer contains an overview of the functionality to be provided and of the data involved. A system architecture is developed that defines the modules of the application system and its dependences. Modules are described by the tasks (either algorithmic or interactive) they have to support and the data that is required or provided. The result is a general view of the system's structure and behaviour.

Action layer: In the action layer a more detailed view of the functionality and the participating data is realised. It contains a skeleton specification of the data structure, views, workflows and HC interaction scenarios. These provide an insight into the functioning of the information system without, however, going into the smallest detail.

Conceptual layer: The conceptual layer contains a complete and detailed design of all the aspects of the information system under development. Data, views, algorithms and HC interactions are described in all the details required to allow for an immediate physical realisation.

Implementation layer: The most concrete layer of this model involves the physical realisation of the application system so that it is executable in a computer infrastructure. Data is represented in a physical data model, external views are defined, programs are coded in a programming language and HC interaction is implemented and/or generated.

The implementation layer also involves a thourough testing of the application system to ensure its robustness and correctness. When the system is in operation, extensions and modifications of the system usually are often required to correct errors or meet new requirements.

The design layers of the alCoD model define certain *perspectives* on the development state of the design. Using abstraction and refinement, it is possible to move from one layer to those layers lying above and below, respectively.

Design strategies. In principle, the alCoD model includes similar phases like those that are defined in software engineering in general [PB93, Som95]:

- problem analysis and planning,
- requirements specification,
- system and software design,
- implementation and component testing,
- integration and system testing and
- operation and maintenance.

These phases have been defined to allow for a planning, scheduling and guiding of the process of software development. A number of phase models have then suggested how to map these phases onto concrete stages and how to sequence them. E.g., Figure 5.2 shows the classical waterfall model [Roy70] which combines the first two of the above phases into one stage and proposes to step from each stage forward to the next one. It also allows for backward movements in the case of errors or inconsistencies that are discovered in later stages.

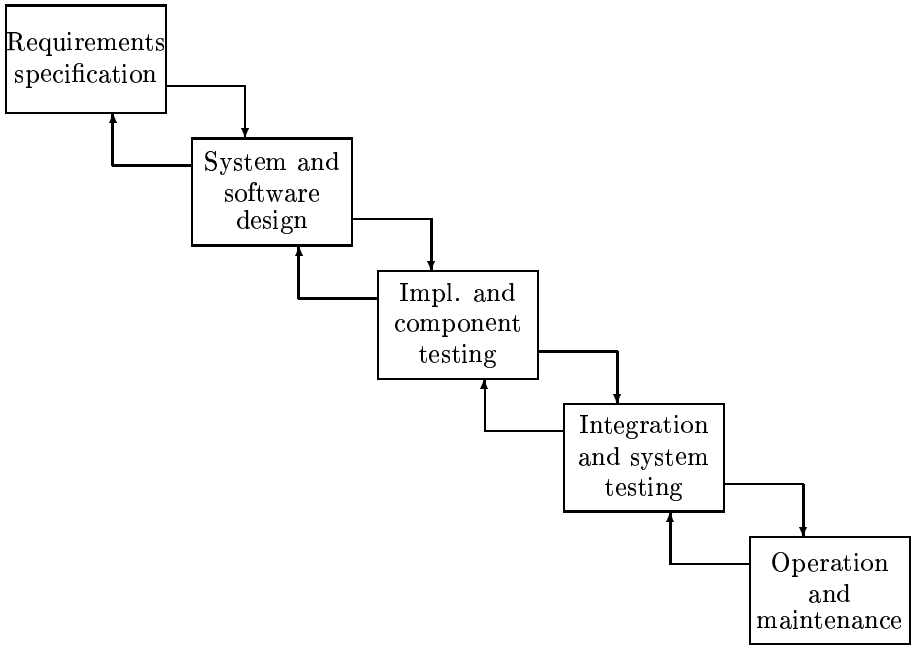


Figure 5.2: The Waterfall Model

On its introduction, the waterfall model was received with great enthusiasm. Later on, it has been criticised for its inflexibility: Because it requires a complete processing of each stage, it is impossible to leave parts of individual stages unfinished for the sake of getting an early overview of the entire system. Other models like various prototyping models or the spiral model by Boehm [Boe88] have, therefore, been proposed.

Figure 5.3 shows an illustration of Boehm's spiral model. It consists of four phases that form a loop. The *objectives* phase in the top left corner involves an analysis of the tasks to be fulfilled and of the resources available. In the *evaluation* phase in the top right corner, the alternatives are evaluated and one of them is realised in the form of a prototype. This prototype is then analysed and tested in the *verification* phase in the bottom right corner. Based on this analysis further modifications, extensions, re-developments, etc., of the prototype are decided upon in the *planning phase* in the bottom left corner. This process is repeated until a satisfying system has been developed.

The early existence of prototypes is one of the main advantages of models of this type. Prototypes can be used to enhance communication with customers and to provide early feedback to the system developers.

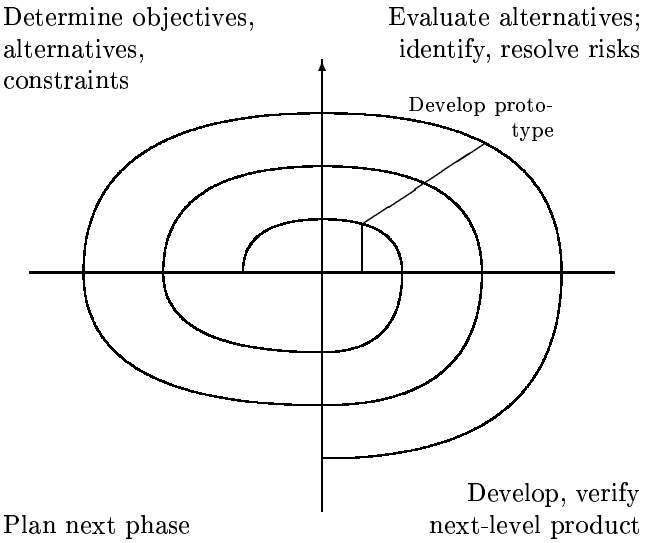


Figure 5.3: The Spiral Model

Design strategies in the alCoD model. The focus of the alCoD model lies on the support of an integrated development of different design aspects (data structure, views, behaviour, HC interaction). It does not enforce particular design strategies. In fact, different phase models can be superimposed on the alCoD

model so that different design strategies are realised.

A design strategy starting at the motivation layer and then stepping down the abstraction hierarchy with only minimal backtracking in case of the discovery of incompletenesses or inconsistencies, would correspond to the classical *waterfall* approach. A proceeding that loops iteratively through the individual design layers and uses prototypical (and also partial) implementations for an evaluation and further refinements and modifications, realises a *spiral* approach. It is also possible to use the alCoD model in a *bottom-up* fashion, i.e., to start on any of the less abstract layers and to define more abstract views and motivations subsequently. A bottom-up approach could be applicable in easy-to-overlook problem situations where a thourough structuring of the domain and possible solutions is a task of little complexity.

Various other design strategies are conceivable. Which one is selected depends on the complexity of the problem and the application domain as well as on the organisation of the design team and on the expertise, capabilities and preferences of individual designers [BCN92, SSR93, Ape94].

5.2 Dependences between Design Aspects

It has already been mentioned that the design of HC interaction on the one hand and the design of database views and the overall system behaviour on the other hand are interrelated. In this section we will examine the dependences between these design aspects more closely. A theoretical model for such dependences (cf. Appendix A) is currently being developed [Cla00].

The design of HC interaction gains from the Co-Design model in so far as the Co-Design model defines inter-dependences between design aspects that can be exploited for the facilitation and improvement of the entire design. The links between HC interaction design, view design and behaviour design lead to the following consequences:

- Relation between HC interaction and view design
  - Interaction processes consume and produce data. This data will be extracted from and stored to the database which can be achieved by means of external views.

    In the terminology of the theoretical model presented in Appendix A this means, that in the context of the category $\mathcal{LIS}$ (the locally observable information system), the definition of morphisms (interaction processes) also leads to the definition of objects (external views) that are arguments and values of those morphisms.

  - For views it has to be decided whether or not they are to be made available to the user. If yes, corresponding interaction processes have to be defined.

    According to the model this means that the definition of objects (external views) may lead to the definition of morphisms (interaction processes).

- Relation between HC interaction and behaviour design

  - Interaction processes are part of the overall behaviour. The definition of interaction processes and knowledge about the objects participating in HC interaction, can lead to the definition of supporting processes for, e.g., database access, necessary manipulations of data, etc.

5.3 The Development of HC Interaction in the alCoD Model

HC interaction is one of the design aspects identified in the Co-Design method. In the following, we will explore two issues: firstly, the HC interaction design tasks on different design layers; secondly, mutual dependences between HC interaction design on the one hand and the neighbouring aspects of view design and behaviour design on the other hand.

A motivation: Screenplay writing. Some of the terminology used in the following has been borrowed from screenplay writing [Val82]. Similarly to HC interaction design, screenplay writing has the task to develop a strategy for communicating about a series of activities (a story) to an audience. Cooperativeness is essential both for HC interaction design and screenplay writing. Excepting certain avant-garde productions, screenplays too have to ensure that the audience is able to follow the story and individual scenes, that inconsistencies are avoided, that the aesthetical taste of the audience is met, etc. Unfortunately, screenplay writing offers little support for the abstraction from particular interaction situations since it is well able to focus on one medium and a certain audience type only.

Of course, screenplay writing is a creative process, guided by inspiration and many spontaneous ideas. Generally, however, it comprises the following phases:

Exposé: Screenplay writing starts with the finding of a *topic*, e.g., the story of a friendship, the events of a revolution, etc. A number of facts around this topic are described like the living conditions of two people, the way they meet and the events that make them friends, or the events leading up to

a revolution, the occurrences of the revolution itself and the aftereffects. These facts are established to allow for and guide a later narration of the entire story.

Adaptation: The facts are then reduced to a simple, clear *story* line. A first skeleton of the story is developed.

Treatment: The story is told without too much detail. A *scenario* is developed that includes individual scenes and their sequencing.

Screenplay: The *scenes* are elaborated in detail. Instructions for actors, camera positions, lighting, etc., are included. The result is a complete description of the story that can be used for its physical realisation, the shooting.

Shooting: Based on the screenplay, the movie is then realised. All the participants act according to the instructions of the screenplay, thus creating a *materialisation* of the screenplay.

Screenplay writing can serve as an interesting inspiration for the assignment of HC interaction design tasks to the individual layers of the alCoD model—even if it does not considerably contribute to the abstraction required to support interaction in heterogeneous and dynamic environments. HC interaction design takes a similar path of development: from the formulation of a goal or a topic to the development of a rough outline of the story, i.e., the task structure of the application system; scenarios are then defined for the support of those tasks, and eventually the scenes that make up scenarios are described in full elaboration; this scene description—along with the scenario structure—is then used to generate HC interaction.

In the following, we will describe HC interaction design tasks on different design layers in more depth. Along with issues that directly concern the modelling of HC interaction, we will also set forth the development of models of the interaction environment and we will point out links to the design of views and the overall system behaviour. The running example of a travel information system accompanies and illustrates the issues described.

5.3.1 The Motivation Layer: Topics

In the motivation layer, we define the *topics* of the information system to be developed, i.e., we basically answer the question *What for?*—generally in an informal manner. The motivation layer includes a description of the area of application, the goals and intended effects, directions for the development and a characterisation of the resources available. A feasibility study usually accompanies this development phase.

General statements are prevailing in the motivation layer. A clear distinction between the individual design aspects of data structure, views, behaviour and HC interaction is usually not desirable nor even possible.

Following is a description of the individual components of the motivation layer. Example 5.1 serves as an illustration.

Application area: First of all, the motivation layer defines the area and domain the information system has to operate in. Here we describe

- the *type* of application.
- the application *focus*, i.e., interactive tasks users might be interested in and which the information system has to support,
- the targeted *audience*, i.e., user groups or even technical equipments.

Goals and effects: Furthermore, it must be stated what is expected of the information system. This concerns the design result itself as well as the effects of its eventual operation. Such statements help directing the process of development and serve as a measurement of success later on.

Directions: At the start of the development process we can also determine certain directions, guidelines and defaults for the expected result. These can concern the appearance of the system, the amount and type of data to be stored and provided and the kind of functionality and interaction to be offered. We can also state directions for the process of system realisation like the existence of a prototype after a certain period of time. Directions can be given absolutely or in comparison to other systems.

Resources: Finally, the required and the available resources have to be analysed: costs, time and human resources. A *feasibility study* accompanies the entire process of motivation definition. It helps to estimates the financial success of the whole enterprise and to find trade-offs between what is desired, what is required and what is possible. The feasibility study shows that interdependences between the individual aspects of the motivation layer are unavoidable: high costs might, e.g., make a higher profit necessary; if it is found that a certain profit cannot be exceeded, feasibility might not be within reach, etc.

The motivation layer is characterised by relatively general statements. Some of these statements hint, however, at aspects that concern the HC interaction of the system. The development of an abstract HC interaction model will, e.g., be guided by statements concerning the application focus (interactive tasks) and by various directions concerning data and functionality to be provided. The targeted audience is the starting point for social and technical environment models,

and directions concerning the appearance will often be reflected in the stylistic environment model.

Example 5.1: Travel information system: Motivation layer

Application area: Travel agencies

Type: Travel and tourism information system

Focus: Browsing/searching the database, bookings and payments, travel assistant for travel preparations

Audience: Young to middle-aged customers, independent travellers (backpackers, individual travels);
English speaking, yet country specifics in appearance and offers

Goals and effects:

- Win the yearly design contest.
- Achieve turn-over of 500.000 in the first year.
- Gain 10% market share within 5 years.

Directions:

Content: Good-value offers, no ship cruises, sport

Behaviour: Allow travel assistant for registered users only;
include short cut to special offers wherever possible

Appearance: Colours: green-orange

Development process: First prototype after 6 months

Resources:

Budget: 1.000.000

Time: 1 year

People: 2 application designers, 3 programmers, 1 media designer

■

5.3.2 The Business Process Layer: Stories

The business process layer is responsible for describing the *What?* of the design. Here we develop the *stories* of HC interaction, i.e., we describe and relate the interactive tasks (from the application focus) that are to be realised. The business process layer thus includes interactive tasks along with supporting processes and participating information. Environment models also begin to play a role. Example 5.2 illustrates results of the business process layer.

Interactive tasks: Interactive tasks correspond to intentions that human users might have when using an information system. Naturally, such tasks depend on the application domain in question.

Those intentions that are to be supported by the information system are the stories to be realised. In contrast to a screenplay, an information system can have several stories. It has as many stories as there are ways and opportunities of using it. The user might even discover stories that had never been thought of by the developers.

Interactive tasks are refinements of the application focus as described in the motivation layer. In the remaining development process, stories can either be refined individually or in combination.

Information and processes: Interactive tasks are generally connected to information that is to be exchanged and to processes that serve as a support for interactive tasks. *Information* naturally plays a role in interactive tasks as HC interaction is about providing information to the user and extracting information from the user. Information can be associated to the transitions between interactive tasks and to start and end points. It describes data that has to be fed into interactive tasks and that is the outcome of interactive tasks, respectively. It will eventually serve as input for the development of views. *Processes* can be responsible for retrieving data from the database, for storing data in the database and for calculations. They are to become an additional part (along with mere interaction processes) of the overall system behaviour.

Environment models: When stories are defined, different user types and preferences have to be considered too. Certain interactive tasks or transitions might be available for certain users only or different alternatives could be preferred. Such requirements lead to the incorporation of the distinguishing property into the appropriate environment model.

The result of the business process layer is a graph connecting the interactive tasks to be supported with a description of the participating information, supporting processes and relevant environmental factors.

Example 5.2: Travel information system: Business process layer

Interactive tasks: (cf. Figure 5.4)

sequencing (and possibly extension) of interactive tasks as first suggested in the motivation layer;
branches and alternatives;
compulsory and mandatory transitions

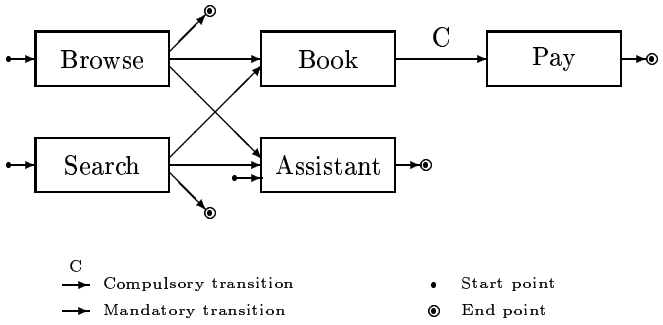


Figure 5.4: Example: Business Process Layer

Information:

- tour information (to start browsing and searching)
  - flights
  - accommodation (hotels, hostels, camping)
  - day trips, guided tours, entertainment, talks
  - car rental, bicycle rental
- tour offers (to start booking and travel assistant)
  - offers selected by the user from tour information
- travel assistant info (to start travel assistant)
  - immigration requirements (visa, vaccinations, etc.)
  - country customs and information (climate, landscape, etc.)
  - check list for managing absence
  - advertising for travel companions, neighbourhood help (watering the flowers, feeding the dog, etc.)
- client information (to start payment)
- bookings (outcome of booking, to start payment)

- payments (outcome of payment)
- search information (to start searching and travel assistant)

Processes:

- database access (retrieval, storing)
- checking of payment

Environment models:

- Social model:
  - registration (for access to travel assistant)
  - home country/region (to target travel assistant advertisements)
  - duration of journey (to adjust check list)
  - age, interests, income (to target special offers)
  - preferences and experience (searching vs. browsing)
- Environment model:
  - interface system (to generate appropriate presentation)
  - audio equipment (if country information includes playing of national anthem)

■

5.3.3 The Action Layer: Scenarios

In the action layer, the stories of the business process layer are underlaid by more concrete *scenarios* that answer the question *How?*, i.e., how to realise the stories. The result is a pre-design, a design that captures all the prominent aspects of the information system without going into every detail.

In the action layer, interaction steps and relations between interaction steps are defined that realise the individual stories. The supporting processes and participating data are refined, and the influence of the environment models is further specified.

Interaction steps: Interaction steps describe the activities that play a role in the fulfillment of interactive tasks as specified in the business process layer (see above). Interaction steps are responsible for the provision of information to the user and/or the acquisition of information from the user, e.g., they can be used to simply display information or to simply have information entered or—in combination—to display information and to have other information entered on the basis of the displayed information.

Information and processes: The action layer pre-design also describes information that serves as input or output to interaction steps as well as supporting processes that perform the database access or necessary transformations of the participating information.

The specification of the required information goes along with the development of a view skeleton in the view design aspect. The entirety of interaction steps and supporting processes forms a subset of the overall system behaviour. For the behaviour aspect of the information system, other processes need then to be specified that are not of direct relevance for the HC interaction component of the system (such as statistical analysis, garbage collection, etc.) but that are an integral part of the system functionality.

Transitions between steps/processes: In refinement of the graph that represents interactive tasks, interaction steps, supporting processes and participating information are combined into an *HC interaction skeleton*. The skeleton is a bipartite hyper-graph [CLS99b] with interaction steps/supporting processes and information that are connected to each other by (possibly branched) transitions.

The HC interaction skeleton shows how interactive tasks are to be realised, i.e., it contains and relates the steps and processes required. In order to fulfill a task, different strategies can be used, i.e., different steps, different sequences can be chosen.

Environment models: As far as differences in the task fulfilling or in the participating information are motivated by different user types, technical requirements, etc., these conditions are used to extend the structure of environment models.

Example 5.3: Travel information system: Action layer

HC interaction skeleton: Figure 5.5 shows an example of an HC interaction skeleton. It combines a number of interaction steps, supporting processes and participating information into a bipartite hyper-graph. The labelling of the branches corresponds to the association semantics presented in Section 4.3.

E.g., the supporting process *Select info* which is responsible for the search for travel offers can be started in two ways: firstly, by the availability of *Travel info* (content of the travel database) and *T-travel info* (search criteria given by the user) as input data; secondly, by the additional availability of an already existing selection (*choice*) that serves as the basis for a further refinement of the result.

These processes realise the interactive tasks of search, browse, book and pay as indicated in Figure 5.5.

Information: Information participating in the HC interaction skeleton can be specified by more or less complete type description as has, e.g., been shown in Figure 4.2 on page 82. These type descriptions capture the structure of information. Figure 5.5 shows which information is relevant in our example.

Environment models: Environment models could be *extended* by the following properties:

- Social model: intentions of user (to more or less prominently offer different transitions, not all users are interested in booking)
- Technical model: printing facilities, secure connections (to choose between online printing and surface mailing of travel documents)

■

5.3.4 The Conceptual Layer: Scenes

In the conceptual layer a *detailed design* of all interaction-related aspects according to the ACE meta model is realised, i.e., individual interactive steps are extended into detailed *scenes*. After answering the *What for?*, the *What?* and the *How?* we now answer the question *How exactly?*. This includes

- the definition of all interaction processes in refinement of interaction steps,
- the specification of all participating data,
- the inclusion of all supporting processes (whose detailed specification is, however, part of the overall behaviour design),
- the building of a bipartite hyper-graph that connects data and processes,
- the specification of data characteristics and relations,
- the specification of process characteristics and relations,
- the conditioning of transitions, characteristics and relations and
- the building of the corresponding environment models.

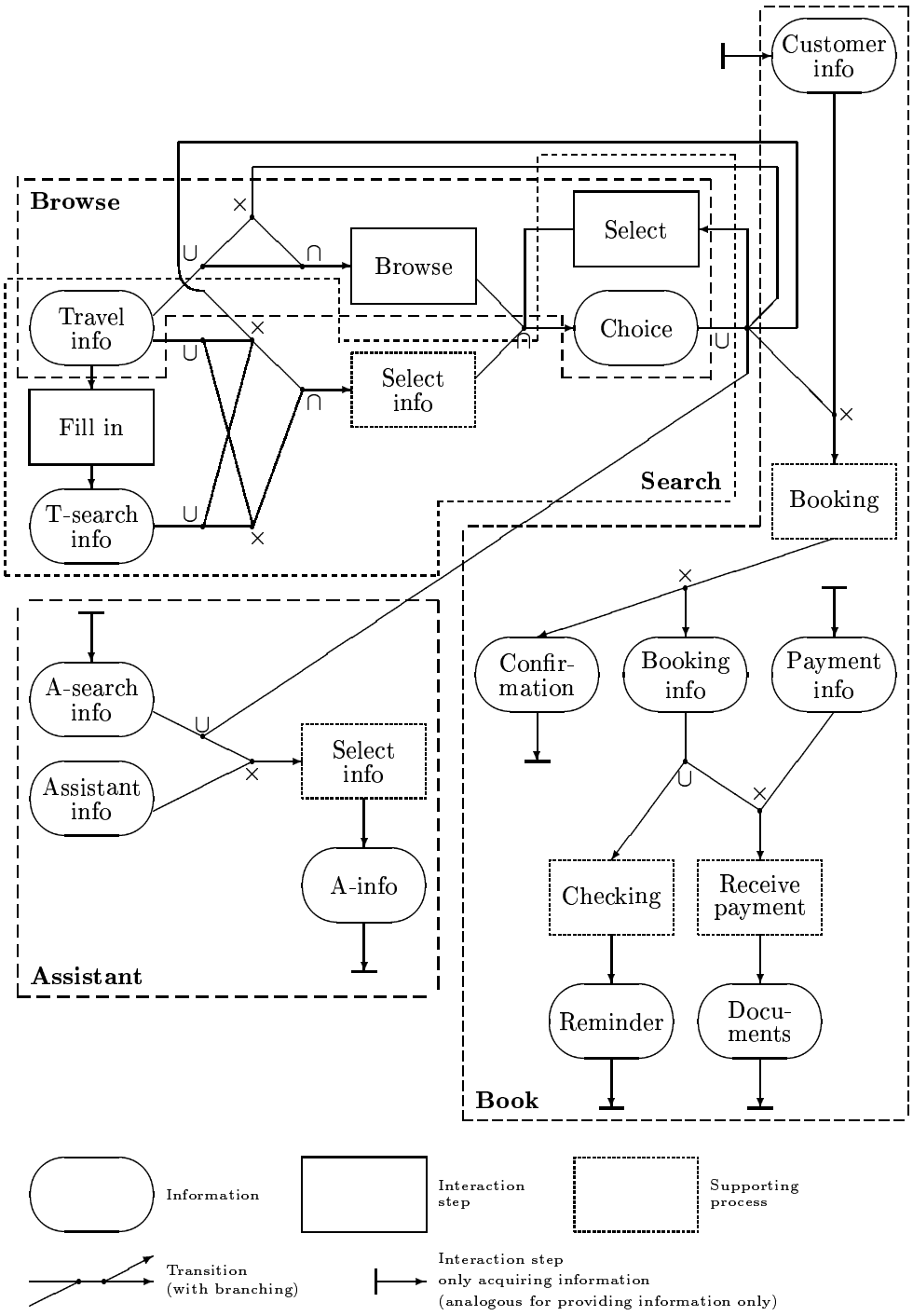


Figure 5.5: Example: Action Layer

Figure 5.6 shows an abridged example which is part of an abstract interaction model in a graphical notation. The example corresponds to the provision of T-search info with the help of the content of the database (Travel info/T-info in the example). We assume the existence of hotel information only to improve the readability of the example.

First of all, help information Help is provided. The individual components of this data type are linked to the components of T-search info by help relations. Moreover, the components of T-search info are associated with emphasis characteristics (E), relevance characteristics (R), default relations and elaboration relations (Elab). In the case of the emphasis characteristic of the price component we have illustrated the use of conditions: For student users emphasis is set to a high value, for other users to a lower value. We have also included a computational process Compute that is responsible for the calculation of price categories which are then used as suggestions for the price component of T-search info. Again, different categories can be calculated based on the status of the user (the respective conditions have been omitted here). In these conditions, some of the content of the various environment models is reflected.

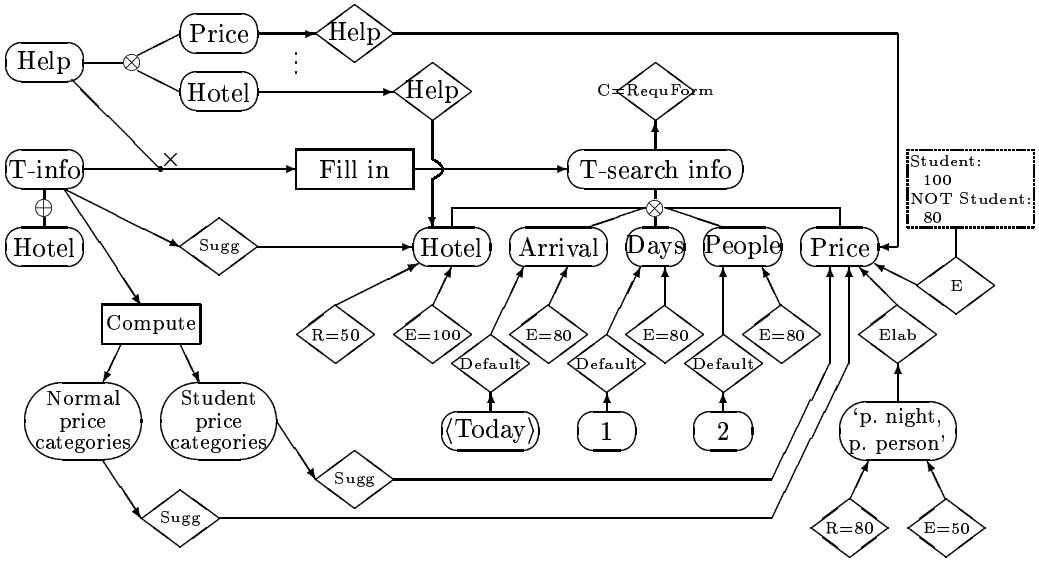


Figure 5.6: Example of an Interaction Specification

5.3.5 The Implementation Layer: Materialisation

Now that we have specified what to do and how to do it, it can be done. In the implementation layer, the conceptual model is *materialised*, i.e., it is transformed

into an executable form. Principally, there are two possibilities to achieve a materialisation:

- The conceptual model can be considered as a guideline for a manual implementation.
- Alternatively, the conceptual model can be used as the basis of an automatic generation of HC interaction.

In both cases, guidelines that have been established in the motivation layer influence the eventual outcome. The content of environment models can be used to achieve a customisation of HC interaction.

This work focuses on the second approach of automatic HC interaction generation. It proposes the use of rules and interface drivers for the transformation of the conceptual design. The advantage of such an approach is its flexibility: changes in the conceptual design are automatically reflected in the implementation without undergoing another manual re-implementation process.

A detailed account of interaction generation mechanisms is given in Chapter 6; its practical realisation is illustrated in Chapter 9. The strategy that is being advocated in this work includes the definition of *rules* that determine how constructs of the conceptual model are to be mapped onto constructs of particular interface systems. *Drivers* then apply these rules and generate the actual interaction at system run time.

5.4 Actors in Human-Computer Interaction Development

The development of HC interaction involves several tasks with different foci: application functionality, screen design (guideline specification) and rule and driver implementation. It must be recommended—for efficiency reasons—to entrust these tasks to experts in each of the fields concerned. Ideally, HC interaction design involves the following experts:

Application designers are responsible for the development of the abstract conceptual HC interaction model, i.e., they specify interaction functionality (by means of data descriptions and processes) and meaning (by means of characterisations and relations).

Interface designers determine the appearance of HC interaction by developing guidelines for the translation of the constructs of the HC interaction meta model into physical presentations.

Programming experts transform these guidelines into rules and implement drivers for rule application. These drivers then perform the automatic generation of HC interaction.

To support the involvement of different experts, a good project management is required that defines communication and work strategies, responsibilities and checkpoints [BB97, Ste99]. In HC interaction development, it is especially important to encourage and facilitate communication

- between application designers and interface designers on the *meaning of modelling constructs* and
- between interface designers and programming experts on the *realisation of design guidelines.*

Presentation control by guidelines. Guidelines are a critical point in HC interaction development. By means of guidelines the presentation of HC interaction on the screen, in a natural language interface, etc., is determined, i.e., for the individual objects used in the conceptual model, rules determine the constructs to be used, their visual/auditive properties, their arrangement, etc. Rules can originate from

- the interface designers,
- stylistic defaults specified by the customer of the development process (motivation layer) or
- individual users once the information system is in operation (self-customisation as far as it is allowed by the implementation).

It is unavoidable that these specifications occasionally contradict each other. Strategies have to be developed that can resolve such contradictions. Possible strategies are:

Order of rule application: The order of rule application determines the eventual presentation settings.

Prioritisation: Rules are given priorities. Values of presentation settings are stored along with the priority of the respective rule. Only rules with a higher priority can overwrite the presentation settings.

Chapter 6

Automatic Generation of Human-Computer Interaction

In the previous chapters we have considered the conceptual modelling of HC interaction. We have introduced a number of abstract constructs for interaction modelling (cf. Chapter 4) and we have illustrated the process of interaction modelling (cf. Chapter 5). The result of the conceptual modelling process is an abstract model of HC interaction that can now be used for concrete materialisations—materialisations that are customised to the social and technical features of interaction environments, to stylistic guidelines and to the current state of interaction.

In the following, we will present approaches to materialisation that are based on automatic generation of HC interaction. In Section 6.1 we will describe the principles that have to be observed in interaction generation. Section 6.2 introduces the concept of interaction objects which is central to the process of automatic HC interaction generation. Section 6.3 presents an architecture for automatic interaction generation. Finally, in Sections 6.4 and 6.5 we will develop a case study of possible concrete evaluations and presentations of various abstract interaction modelling constructs in different interaction situations, i.e., depending on the social, technical, stylistic and interaction factors of the interaction environment.

6.1 Principles of Automatic Generation

Interaction must follow certain principles in order to be successful. In Section 1.4 we described Grice's *conversational maxims* that formulate the basic requirements for interaction in general [Gri75].

Successful interaction is one of the key issues to make an information system successful in general—besides other criteria like quality and amount of information and a robust and appropriate functionality. Interaction gives access to both

information and functionality—the most abundant information and functionality are useless without appropriate and usable interaction mechanisms that make them available to human users. Interaction, therefore, is a truly critical aspect.

¿From the conversational maxims, we can derive describe principles of quality that characterise the goals of good and successful interaction materialisation. Any interaction materialisation (be it manual or automatic) should aim at pursuing these goals in order to render HC interaction successful.

Understandability: First of all, the user needs to be able to interpret and to correctly interpret the contribution of the information system, i.e., to read the information, to use the functionality, etc. This is achieved by appropriate *quantity* (sufficiency of information), *relevance* (avoiding illogical jumps) and *manner* (clarity of expression and structure).

Ergonomic quality: HC interaction is facilitated by interaction means that adhere to common ergonomic style guides. Ergonomic style guides are formulated based on knowledge of the human cognitive process and help to ensure an appropriate *manner* of interaction. An ergonomic layout, colour code, metaphor usage, navigation, etc., help to greatly enhance the usability of physical interaction.

Motivation (appeal): Motivation is a very general goal. It actually includes understandability and ergonomic quality. It often is, however, yet a little more.

Motivating interaction should make the user want to repeat his/her experience. In terms of interaction and in addition to the criteria of understandability and ergonomic quality, motivation is achieved by using those interaction mechanisms that are most appealing to the actual user.

There are, e.g., several colour codes that can be used for interface design, colour codes that combine matching and appealing colours. Only certain colour schemes, however, may be motivating for a young and 'trendy' audience. Such an audience would prefer—and thus be motivated by—a colourful and maybe to some degree even disharmonious interface whereas such a choice would rarely do with other types of users.

Customisation: This leads to the next goal, namely customisation. Customisation is essential to meet the requirements of different interaction contexts and of different users and the technical equipment they use.

What is understandable, ergonomic and motivating in one situation does rarely apply to all the others. It has, therefore, to be determined individually how the above criteria can be met in a specific situation.

6.2 Interaction Objects in the Automatic Generation of Human-Computer Interaction

Before presenting an architecture for automatic interaction generation we will now explore the notion of interaction objects which are the central concept of the generation process.

The generation of HC interaction aims at the support of specific interaction instances (cf. Definition 7 in Section 1.5). An interaction instance according to that definition is described by a number of transformations (i.e., interaction and computational processes) where an interaction partner consumes interaction entities (i.e., data) and produces interaction entities (data) which are then available to other interaction partners. In a concrete *realisation*, those transformations that concern interaction processes are represented by *interaction objects*.

Definition 8 *An* interaction object O_I *is a collection of*

- *an input data type,*
- *an input data value,*
- *an output data type,*
- *characteristics of input data and output data and*
- *relations between data.*

The automatic generation of HC interaction as it is discussed in this work, involves, firstly, the *run-time generation* of interaction objects and, secondly, their *physical presentation*.

6.2.1 The Generation of Interaction Objects

Interaction objects represent what is to be communicated between the information system and its human user at a specific point of time. Interaction objects are generated at system run time to allow for a fine-tuning of the *time* of their generation as well as their *content* to the actually encountered situation.

In the following, we will explore how the generation of interaction objects is invoked and how the content of interaction objects is constituted. Interaction objects can be generated by means of *evaluation rules* as described in Section 6.3. In Section 6.4 we illustrate the influence of the interaction environment on the generation of interaction objects.

The syntax of interaction objects corresponds to a subset of the abstract HC interaction modelling language: only data and their characteristics and relations.

Invoking the generation of interaction objects. The generation of interaction objects is tied to the execution of transformations, i.e., of interaction processes in our model. Interaction processes can be invoked on the user's or the system's initiative.

- A user may start an interaction process deliberately, i.e., he/she may be presented with a list of interaction processes (all processes, all processes which are currently executable, etc.) and can select one process for execution.
- The system itself can start an interaction process based on a more or less sophisticated planning of the interaction flow. A simple way is to continually check the executability of all processes and to start any executable interaction process randomly, based on its waiting time, based on the working load of the processor, etc. A more sophisticated approach to the planning of the interaction flow can take into account
  - the qualitative characteristics of processes, e.g., to select processes according to the purpose of the user's visit (only browsing but no booking processes) or according to known user preferences (first book a flight, then a hotel),
  - the quantitative characteristics of processes, e.g., to execute processes with high priorities first, and
  - organisational relations of processes, e.g., to execute grouped processes without interruptions and to execute sequenced processes as specified.

If still necessary, the executability of the transformation is checked before the generation starts. The checking of the executability corresponds to the application of the satisfiability function *Sat* (cf. Section 1.5). In the case of the language presented in Section 4, the satisfiability function contains

- the checking whether all the required input data is available and
- the checking whether the current user is entitled to perform this interaction process.

Constituting interaction objects. Once it is decided that an interaction process is to be performed, a corresponding interaction object is generated. This involves the following:

- The input data that has been selected as being required is *retrieved*. The retrieval can, e.g., be based on the view definitions that were developed in parallel with HC interaction design.

- It is decided *which output data is required.* If the transformation was invoked because some output data was required due to the planning of the system, this is already known to some extent.

  Part of the output data can be information on how to proceed further, i.e., to which following process this output is to serve as an input (association semantics: + (disjoint union), cf. Section 4.3).

- Relevant *characteristics* concerning the visibility and the presentation of data as well as *relations* concerning their arrangement and integrity constraints are selected.

It is moreover possible to split or to combine interaction objects. Splitting involves the division of the participating data along with the associated characteristics and relations into two or more (not necessarily disjunctive) sets. Combination means the join of the data, characteristics and relations of several interaction objects into one common object.

An interaction object can be *split* if, e.g., space restrictions for the interaction medium or the interaction channel exist so that all the required information cannot be transported at once. If the current user has restricted cognitive abilities or if he/she simply prefers interaction in small quantities, the splitting of interaction objects can also appear appropriate. To decide on how to split interaction objects, it is above all the relations between participating data that can be exploited: It is, e.g., possible to separate data that is related by narrative relations, i.e., data that serves as explanation, help, summary, etc. This information could be made available in separate interaction objects on the user's request. Organisational relations can be exploited in so far as data which is sequenced can be distributed into different interaction objects which are sequenced accordingly and data which is grouped should not be split.

To *combine* interaction process is more difficult to decide on, especially by automatic means. If it is still to be realised, groupings of interaction processes can suggest a combination of the respective interaction objects. Moreover, organisational relations that associate data which participate in different interaction objects can make a combination appear reasonable.

6.2.2 The Physical Presentation of Interaction Objects

An interaction object represents what is to be communicated at a certain point of time. It reflects, so to speak, the *message* of interaction. An interaction object must then be given a physical presentation, i.e., it must be displayed on a screen, must be translated into a natural language utterance, etc.

The physical presentation of interaction objects involves *(a)* the selection and generation of interface constructs and *(b)* the assignment of construct settings

and general interface settings. The presentation of interaction objects can be performed by means of *presentation rules* as is described in Section 6.3. Examples of the influence of the interaction environment on the presentation of interaction objects are given in Section 6.5.

Selection of interface constructs. After the selection of relevant concepts from the HC interaction model into an interaction object, these constituents are translated into constructs of the current target interface system which can be a Web browser, any other graphical interface system, a natural language interface, a telephone display, etc. Every interface system makes a number of constructs available that can be used to build interfaces. In a graphical interface system, common constructs are windows, text fields, scroll lists, buttons, etc. [Hel91, Bra92]. Hypertext systems have additional constructs like links and anchors [KPL95].

Interface constructs are chosen to present the data of the interaction object. Usually this can be achieved by mapping data types onto certain interface constructs, e.g., STRING onto text fields in graphical interface systems or utterances in natural language systems. The existence of conversational relations between data can lead to the combination of several data types into one interface construct, e.g., the output data type STRING and an input data also of type STRING that is related to the output data as choice can be translated into a combo box.

Occasionally, several possibilities to translate one data type exist. In such a case, the actual translation can be selected according to the preferences and expectations of the current user.

Selection of interface and construct settings. The overall interface as well as individual constructs often are associated with several settings such as background colour, text colour, size, length, font (for graphical interfaces), volume, modulation (for natural language interfaces), etc. Settings are described by the property concerned (e.g., font) and the actual value (e.g., Times New Roman).

Settings of the overall interface and of individual constructs are above all controlled by stylistic directions as they are specified in the stylistic environment model. As far as these directions leave some freedom, however, settings can again be selected according to the user's preferences and expectations to create an understandable and motivating interface. The selection of appropriate settings is facilitated by the characteristics given in the abstract HC interaction model that refer to the relevance and emphasis of data items, etc.

For the selection of settings, rules can be employed that firstly specify which characteristics are mapped onto which property (e.g., emphasis onto colour) and that secondly specify which values of the characteristic are mapped onto which setting values (e.g., if emphasis greater than 80% then colour is red otherwise

black). Examples of this approach are the Iconographer/Representer system [DW91, Gra97] and the libViewer [RB99] (cf. Section 2.4).

6.3 An Architecture for the Automatic Generation of Human-Computer Interaction

We will now present an architecture for the automatic generation of HC interaction based on an abstract HC interaction model. The architecture illustrated in Figure 6.1 closely follows the ACE framework that has been introduced in Section 3.

The architecture uses two types of *rules* to transform an abstract HC interaction model into a materialisation: evaluation rules and presentation rules. Evaluation rules realise the generation of interaction objects and presentation rules realise their physical realisation. Rules are applied by *rule drivers* that also use information from the existing *environment models* in order to determine the applicability as well as the concrete outcome of rules.

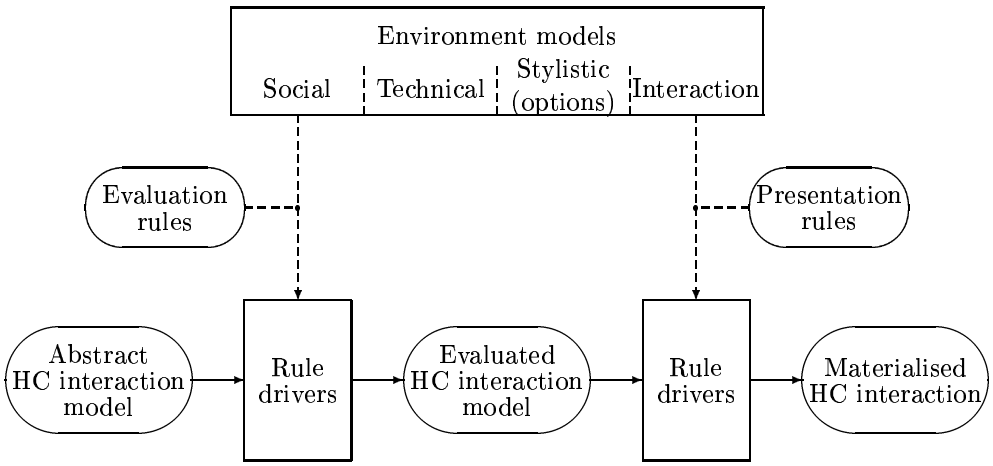


Figure 6.1: An Architecture for the Automatic Generation of HC Interaction

Evaluation rules. Evaluation rules perform a selection or a specialisation of the abstract HC interaction model, i.e., one part of the abstract model is transformed into one or several interaction objects. To that end these rules evaluate the conditions attached to the modelling constructs or the characteristics and relations of data and processes.

By means of evaluation rules we achieve a customisation to the current interaction situation. According to the contents of the social, the technical and the

interaction model, it is possible

- to select interaction processes that are at all executable,
- to select one process for execution,
- to select and retrieve data to be provided,
- to select data to be extracted,
- to select characteristics and relations of data which are relevant for the eventual presentation.

To give some examples, data can be provided according to the user's preferences (hotel information only, but no flights) or expertise (help information for novice users). The technical capabilities of the target system also play a role in choosing, e.g., between text and image data. Some users might not be interested in booking processes while others do not want to be presented with any defaults given by the application system. More details and examples can be found in Section 6.4.

There are two types of evaluation rules:

Direct evaluation rules: Direct evaluation rules work by analysing the *conditions* contained in the model. Their application is straightforward: the applicability of the condition is checked and the construct to which the condition is attached is either selected or excluded.

Indirect evaluation rules: If *characteristics and relations* themselves are to be analysed for the evaluation, then indirect rules are defined independently from the model. These rules contain properties of the model and/or the environment as a rule head and a reaction (selection or exclusion) as a rule body. In fact, these rules realise heuristics. An example of such a rule is the following that excludes all booking processes for interaction situations whose goal is not booking.

```
NOT (interaction_model.goal=booking)
   => EXCLUDE proc p WHERE p.purpose=booking
```

Presentation rules. Presentation rules determine the eventual presentation of interaction objects by means of a translation of the constituents of an interaction object into such constructs as are available on the interface system to be addressed. Presentation rules are defined for

- selecting and generating interface constructs based on types of data,
- selecting and determine settings of the entire interface and of individual constructs based on characteristics and relations,
- arranging interface constructs within a contribution based on characteristics and relations.

Since interaction presentation is basically a translation from the modelling language into the interface language (cf. Section 1.6.2), presentation rules are organised into *rule sets* that are dependent on targeted interface systems. For the presentation of interaction objects, only those rule sets are selected that translate into the interface system in question.

For an illustration of the general format of presentation rules consider the following very simple rule that translates data of the type STRING into a HTML construct:

```
d.type=STRING
   => <FONT>$d</FONT>
```

Environment models. To enhance the flexibility of rules and an easy customisation of interaction generation, rules can operate with *variables*—variables that are contained either in the rule head or in the rule body. These variables then refer to information contained in the environment models.

- The rule head can contain references to all the environment models: social, technical, stylistic and interaction models. Both the evaluation and the presentation can depend on user characteristics, the technical equipment, stylistic defaults (e.g., always offer bookings) or interaction history and state. This has been shown in the above example of an indirect evaluation rule.
- The rule body can only contain references to the stylistic model, thus filling in specific reactions: construct types, colours, arrangement strategies, etc. Following is a more sophisticated example of a presentation rule (for a presentation in HTML) where the colour of an emphasised text d is set to a value specified in the stylistic model:

```
d.type=STRING AND d.emph>80
   => <FONT COLOR="$stylistic_model.emph_colour">$d</FONT>
```

The information contained in environment models can originate from various sources: Entries in the social and the technical model stem from the system (based on observations and reasoning) or from the user (self-customisation); entries in the stylistic model from the interface designer, the system or the user; entries in the interaction model only from the application system. In all cases, the information can be specified either absolutely or in relation to other entries. To avoid unlimited manipulation of models, especially by human users, it is, however, important to define the degree to which certain entries are open to manipulations or even knowledge, i.e., write and read rights have to be set for every potential source (system, designer and user).

Rule drivers. Rule drivers are responsible for the application of rules as well as for the checking of conditions in the case of the evaluation of the abstract HC interaction model. In this context, they also have to handle two problems:

- They have to resolve conflicts between contradicting reactions, e.g., two rules specifying different font colours based on different criteria. Conflict resolution can be achieved by a certain sequence of rule application or by rule prioritisation as mentioned in Section 5.4.
- Relations can span several interaction processes, i.e., they can refer to data items that are associated with different interaction process. For the presentation of such non-local relations it is important that settings used for their presentation at their first occurrence are kept as partial temporal style options for later re-use.

How these problems are handled in practise, depends on the concrete implementation. Concrete implementations can also choose to split the application of rules into several steps. Such an approach is, e.g., used in the prototypical realisation that will be presented in Part III of this thesis. Especially in the case of presentation rules, the use of driver hierarchies can help to reduce translation complexity and to enable the re-use of individual translation steps for different interface systems, e.g., more abstract translations for graphical interfaces systems and a later branching into more concrete translations for specific systems like XView, Motif, Tcl/Tk, etc.

6.4 A Case Study: The Generation of Interaction Objects

The evaluation of an abstract interaction model, i.e., the generation of interaction objects, feeds on two sources:

- conditions that are part of the model itself and are checked by direct evaluation rules and
- characteristics and relations of data and processes which are checked by indirect evaluation rules.

Whereas the first case is very straight-forward (conditions are checked and the associated feature is either accepted or rejected), the second case involves some more reasoning. This section outlines the effect of relevant characteristics and relations on the evaluation phase. Characteristics and relations that need to be considered by indirect evaluation rules are

- data type,
- qualitative and quantitative data characteristics,
- rhetorical and organisational data relations,
- qualitative and quantitative process characteristics and
- organisational process relations.

Process characteristics and relations are considered first in order to decide which interaction process to invoke (Heuristics 9–11 below). After that, on invocation of a particular interaction process, *all* direct input data, *all* output data and *all* associated data characteristics and relations are considered for an incorporation into the respective interaction object as long as the attached conditions are fulfilled. Indirect evaluation rules then evaluate these concepts (Heuristics 1–8 below) and decide to either drop some of them or to incorporate further concepts.

In the following, we will first point out for which effects these concepts are used in the generation of interaction objects. For each concept we will then give examples of heuristics that reflect the influence of the environment, i.e., how a customisation of the interaction object is achieved. As an illustration we will provide an example of an interaction object and suggest a few possible customisations.

Data type: effects the selection of participating data

Heuristic 1: (influence of technical factors)
If data type not representable on targeted interaction medium,
then drop it.
(e.g., data of type image not representable on Lynx web browsers and cellular phones)

Heuristic 2: (influence of technical factors)
If size of data type exceeds size of medium or channel,
then drop it.

Qualitative data characteristics: effect the selection of participating input data

Heuristic 3: (influence of social factors)
If 'sports' $\in$ user's interests,
then select data of category 'sports'.

Quantitative data characteristics: effect the selection of participating data

Heuristic 4: (influence of social factors)
If status of user is student,
then select offers characterised as 'cheap'.

Heuristic 5: (influence of social factors)
If user works under time constraints,
then drop data with relevance below 50.

Heuristic 6: (influence of technical factors)
If size of medium below 160 characters or capacity of channel is below 12KB/s,
then drop data with relevance below 50.

Rhetorical data relations: effect the selection of participating input data, often in addition to the direct input data

Heuristic 7: (influence of social factors)
If user is expert user,
then drop data that is related by help relations.

Heuristic 8: (influence of technical factors)
If size of medium below 160 characters or capacity of channel is below 12KB/s,
then drop data related by help relations, suggestion relations and summary relations.

Organisational data characteristics: Groups of data should always be selected together.

Qualitative process characteristics: effect the selection of interaction processes to be started

Heuristic 9: (influence of interaction factors)
If goal of interaction is booking,
then select booking processes.

Quantitative process characteristics: effect the selection of processes to be started

Heuristic 10: (influence of social factors)
If user is working under time constraints,
then drop processes with relevance below 50.

Heuristic 11: (influence of technical factors)
If capacity of channel is below 12KB/s,
then drop processes with relevance below 50.

Organisational process relations: Grouped processes should always be started without other processes interrupting; a combination of the respective interaction objects is also possible.

In Section 5.3 we gave an abridged example of a partial HC interaction model (cf. Figure 5.6 with most of the type definitions being omitted). This example includes the interaction process Fill in. On invocation of the interaction process Fill in, an interaction object would be created with the following content:

Example 6.1: An interaction object

```
Input data:
  Help: RECORD (Hotel: STRING,
                ...),
  Travel_info: SET (Hotel: STRING),
  Student_price_categories: SET (RECORD (From: REAL, To: REAL)),
  DefArrival: DATE,
  DefDays: INT,
  DefPeople: INT,
  ElabPrice: STRING,
Input values:
  Help = RECORD ('Please, enter the name of a hotel.',
                 ...),
```

```
  Travel_info = SET ('Eden', 'Bologna', 'Loigge'),
  Student_price_categories = SET ((1,30), (31,50), (51, 100)),
  DefArrival = 01/04/2000,
  DefDays = 1,
  DefPeople = 2,
  ElabPrice = 'p. night, p. person',
Output data:
  Search_info: RECORD (Hotel: STRING,
                       Arrival: DATE,
                       Days: INT,
                       People: INT,
                       Price: REAL)
Characteristics:
  Emph (Search_info.Hotel, 80),
  ...
Relations:
  Sugg (Search_info.Hotel, Travel_info),
  Sugg (Search_info.Price, Student_price_category),
  Help (Search_info.Hotel, Help.Hotel),
  ...
  Default (Search_info.Arrival, DefArrival)
  ...
  Elab (Search_info.Price, ElabPrice)
```

■

First of all, the example includes the direct input data Travel_info and the direct output data Search_info. Additional input data has been included based on rhetorical relations and on the information that the user is a student (only student price categories have been selected). Moreover, all the characteristics and relations between the participating input and output data have been incorporated. None of the above heuristics were applied so that the dropping of concepts did not happen. Input data is instantiated based on view definitions whose specification lies, however, in the responsibility of the aspect of view design which is not covered in this work.

Imagine a situation where less information can be fitted into one interaction contribution, possibly because of space restrictions of the interaction medium or because the user works under time constraints. In that case much of the indirect input data, i.e., data related via rhetorical relations, can be dropped in order not to overload the respective interaction object. Applying heuristic 8, a reduced interaction object could be created as shown in the following example:

Example 6.2: A reduced interaction object

```
Input data:
  DefArrival: DATE,
  DefDays: INT,
  DefPeople: INT,
  ElabPrice: STRING,
Input values:
  DefArrival = 01/04/2000,
  DefDays = 1,
  DefPeople = 2,
  ElabPrice = 'p. night, p. person',
Output data:
  Search_info: RECORD (Hotel: STRING,
                       Arrival: DATE,
                       Days: INT,
                       People: INT,
                       Price: REAL)
Characteristics:
  Emph (Search_info.Hotel, 80),
  ...
Relations:
  Default (Search_info.Arrival, DefArrival)
  ...
  Elab (Search_info.Price, ElabPrice)
```

■

6.5 A Case Study: Physical Presentations of Interaction Objects

In the following chapter we present suggestions for the physical presentation of interaction objects. Interaction objects contain a selected number of the abstract modelling constructs that were introduced in Section 4. It is, therefore, these abstract modelling constructs that have to be translated into physical presentations.

Concrete guidelines for presentations will always depend on the application system in question, on the aesthetical preferences of customers, designers, etc.

This chapter will, therefore, be restricted to developing a picture of the possibilities of translations. We will look at two very different types of interfaces: graphical interfaces and natural language interfaces. For each of these, we will define the set of features that can be manipulated (presentational variables) as well as correspondences between abstract modelling concepts and these presentational variables. Some attention is also given to potential influences of the interaction environment.

6.5.1 Presentational Variables in Graphical Design

First of all, we want to start the discussion of different presentations with the introduction of some terminology.

The generation of presentations can be understood as a translation between languages: from the language of the abstract HC interaction model to the language of the targeted interface (cf. Section 1.6.2). In order to be able to talk about translations we first need to know about the languages in question: The language of abstract HC interaction modelling has been presented in Section 4. Every interface system, however, has its own language that differs in terms of constructs available as well as grammar. Mullet and Sano define, e.g., a visual language as

> the visual characteristics (shape, size, position, orientation, color, texture, etc.) of a particular set of design elements (point, line, plane, volume, etc.) and the way they are related to one another (balance, rhythm, structure, proportion, etc.) in solving a particular communication problem [MS95].

Thus, like any other language, a visual language relies on a defined universe of possible signs (vocabulary) and on a set of rules for using these signs (grammar). Ways in which images communicate meaning by using colour, perspective, framing and composition, i.e., by assigning visual characteristics and relations to graphical elements, are also examined by Kress and van Leeuwen [KL96].

Visual variables. To describe individual signs and their relations in a visual language, graphical design uses visual variables: size, value, hue, orientation, texture, shape and position. These variables can be used to assign properties to visual elements, to visually group or contrast design elements, etc. Each variable is characterised by its *length*, i.e., the number of discernibly different measurement levels each can support, and by its supported *scales of measurement*, i.e., whether it permits the visualisation of classifications, orderings or amounts of difference. Figure 6.2 illustrates visual contrasts in size, value, orientation, texture, shape and position.

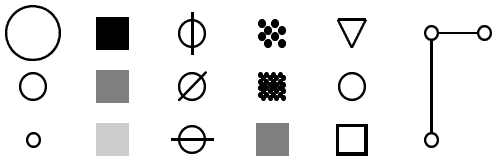


Figure 6.2: Some Visual Variables (according to Mullet and Sano [MS95])

Presentational variables. To generalise this concept, let us call the objects that make up an utterance in any interface language *placeholders*. Just as visual variables describe the properties of elements in graphical design, we can now say that *presentational variables* describe the attributes of such placeholders, i.e., those aspects that can be manipulated, in any interface language. In graphical interfaces we have, e.g., the very indispensable presentational variable widget. The widget variable corresponds to the visual variable of shape. By assigning a widget value to a placeholder, we basically give a form to the placeholder—just as we give a form to natural language placeholders by assigning word values. Other presentational variables for graphical interfaces are size, label, position and reaction.

Each presentational variable is associated with a *domain* which defines the set of possible values. The widget domain can, e.g., consist of the values window, text and button. The values of the position domain are specified in dependence on the respective widget values (or classes). They determine and restrict possible combinations of placeholders, thus defining the grammar of the interface language.

Example 6.3 illustrates the definition of some graphical interface in a very simple language using the presentational variables widget, size, label, position and reaction. Figure 6.3 shows the corresponding interface.

Example 6.3: Defining Presentational Variables

#1.widget = 'window'
#1.size = '250,100'
#1.label = 'Example'
#1.position = '0,0'
#1.reaction = ''

#2.widget = 'text'
#2.size = '50,20'
#2.label = 'Text1'
#2.position = '#1.50,50'
#2.reaction = 'displayHelp(Text1)'

#3.widget = 'text'
#3.size = '50,20'
#3.label = 'Text1'
#3.position = '#1.150,50'
#3.reaction = 'displayHelp(Text2)'

#4.widget = 'button'
#4.size = '50,20'
#4.label = 'OK'
#4.position = '#1.100,20'
#4.reaction = 'cancel(#1)'

■

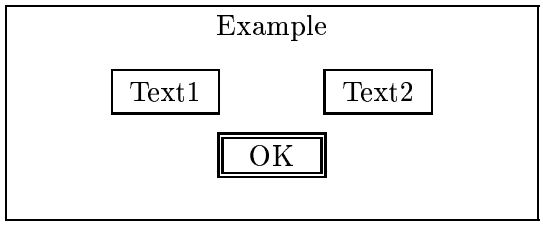


Figure 6.3: Defining Presentational Variables

In the following two sections, we will now define presentational variables for graphical interfaces as well as natural language interfaces. To illustrate the translation between the HC interaction modelling language and interface languages (as described by presentational variables) we will set forth correspondences between abstract HC interaction modelling constructs and the respective presentational variables.

6.5.2 Graphical Interfaces

Graphical interfaces use a number of graphical objects (widgets) for the presentation of information. Graphical presentations are often considered to be especially user-friendly since an understanding of facts can be efficiently supported by the use of visual cues.

Presentational variables. Mullet and Sano [MS95] have emphasised the visual variables *size*, *value*, *hue*, *orientation*, *texture*, *shape* and *position* to describe the characteristics of graphical objects. Figure 6.2 on Page 145 illustrates these (except hue).

The visual variables suggested be Mullet and Sano are, however, not sufficient to fully describe the properties of a graphical interface to an information system. Therefore, we propose the following classification:

- *widget* (shape),
- *label*,
- *action* taken on activation (click, etc.),
- *size*,
- *value*, i.e., 'darkness' or intensity,
- *colour* (hue),

- *orientation*, i.e., display angle, and
- *texture*, i.e., surface appearance.

In the above classification we have added the variables of *label* and *action*—simply to add tags (identifications from the user's point of view) and actions that are executed when the respective placeholder is selected. Since there are several means of selection (mouse click, mouse point, return key, etc.), the action variable can be multi-valued: it can contain several *selection mode–reaction* pairs.

Translating abstract modelling constructs. Table 6.1 shows correspondences between the constructs used for abstract HC interaction modelling which can appear in interaction objects (data and their characteristics and relations) and eventual presentations.

The data type of in information item is usually mapped onto the visual variable *widget* (e.g., STRING onto textboxes, etc.). Qualitative characteristics are translated into the visual variable *label*. This means that, e.g., information categories can lead to selecting particular pre-defined labels (either textual or iconic).

More correspondences are given in Table 6.1 which also shows how the presentation process can be influenced by information on the current interaction environment. E.g., users may prefer textual to iconic widgets or vice versa. Iconic labels can be more appropriate for non-native speakers; textual labels for visually impaired users who could have difficulties in identifying small icons. What has not been included in the table is the influence of the stylistic model which, in fact, can effect any presentational variable by defining global values to be used.

Figure 6.4 shows a possible graphical presentation of the interaction object shown in Example 6.2. Output data have, of course, be translated into entry fields. The respective labels were chosen due to a categorisation of data based on either the respective type or on explicit qualitative characteristics. To each such categorisation different labels can be assigned (in different languages or different icons) that can be chosen from. Defaults and suggestions are directly provided in combination with entry fields, and the elaborating information is made available via a link. The OK button is generated by default to signal the end of user input.

Even for this very simple presentation, variations are conceivable: Iconic labels could be used and the elaborating information on price could be directly included without a link. If applicable, different colours and different font sizes could be used.

6.5.3 Natural Language Interfaces

Natural language interfaces communicate with users by means of natural language—either spoken or written. Often, only a reduced vocabulary and/or grammar is

Construct	**Targeted variable and effects**	**Influence of environmental factors**
Data type	*Widget:* widget selection, e.g., STRING onto text fields, etc.	*User:* preferences for particular widgets
Qualitative characteristics	*Label:* iconic, textual	*User, conventions:* contexts for metaphors, terminology
Quantitative characteristics	*Size, hue, value, position:* gradations of these properties in order to convey differences and scalings	*User:* capabilities for visual distinction (in case of visual impairment), preferences; *User, conventions:* common metaphors
Usage characteristics	*Hue:* dominant colours; *Value, position:* dominance; *Shape:* icons	*User:* capabilities for visual distinction; *User, conventions:* common metaphors
Narrative relations	*Position:* arrangement; *Widget, label:* labels, icons, navigation via links, buttons, etc. *Action:* provision of related data on user request	*User:* preferences; *User, conventions:* common metaphors; *Channel:* avoidance of additional navigation in bad infrastructure
Conversational relations	*Widget:* widgets that can combine input and output, e.g., checklists	*User:* preferences for particular widgets
Organisational relations	*Position:* arrangement; *Hue, value, shape, size:* equality of properties for groupings, gradations for sequences	*User:* capabilities for visual distinction, preferences; *User, conventions:* common metaphors

Table 6.1: Presentations in a Graphical Interface

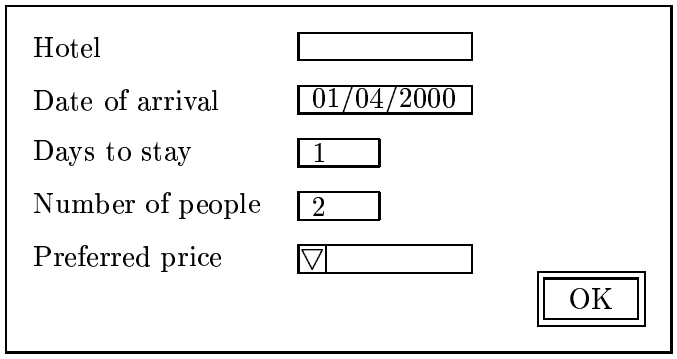


Figure 6.4: A Possible Graphical Presentation

available. Natural language interfaces require little or none experience in the use of computers. That is why they are considered extremely user-friendly. Yet the above named restrictions with regard to vocabulary and grammar often hinder an absolutely natural interaction experience.

Presentational variables. Let us consider spoken natural language interfaces. Natural speech can be characterised by a lot of voice facets; not all of these are, however, suitable for manipulation in a natural language interface. An actor who surprises by switching between different accents, speeds, voice pitches, etc., might be entertaining, but in a natural language interface such changes would be confusing, especially in the attempt to identify the computer as a social actor (which people do as has been shown by Nass et al. [NST94]). We will, therefore, only consider the following properties:

- *lexis*, i.e., the vocabulary (terminology) and phrase types (enumeration, etc.) available,
- *position*, i.e., the position of speech elements within sentences and texts,
- *volume*,
- *melody*, e.g., the typical distinction between statements and questions achieved by raising the voice at the end of questions, and
- *pitch*, e.g., male vs. female voices, etc.

Lexis will be the most widely used property of natural language interfaces since many concepts that can be conveyed by colours, font sizes, etc., in graphical interfaces must be paraphrased in language interfaces.

Note that in contrast to graphical interfaces the *action* variable is not relevant as actions are only taken based on an analysis of the user's entire interaction contribution.

Translating abstract modelling constructs. Possible translations of abstract HC interaction modelling constructs (as contained in interaction objects) into presentational variables of natural language interfaces are shown in Table 6.2.

Naturally, data types are, e.g., matched onto different phrase types such as lists onto enumerations, etc. Again, information contained in environmental models can have considerable influence on the translation. Some examples of this influence are also shown in the table.

Example 6.4 shows a possible representation of the reduced interaction object shown in Example 6.2. This presentation consists of a sequence of questions to the user while the user is expected to answer these questions. The questions themselves could be generated by the simple means of sentence patterns that are filled with parameters [Düs97]. It is easy to see how default and suggestion relations are mapped onto pre-formulated answers.

Example 6.4: A possible natural language presentation

Is there a hotel you prefer?

...

When will you arrive? Today?

...

For how many days will you stay? One?

...

How many people are in your party? Two?

...

Which of the following price categories do you prefer?
Less then 50, 50 to 100, over 100. ...

■

Construct	**Targeted variable and effects**	**Influence of environmental factors**
Data type	*Lexis:* Natural language possesses only one 'widget' type, namely 'word'. So all types are mapped onto words, phrases (combinations of words), etc.	*User:* preferred or known terminology as a subset of the terminology available
Qualitative characteristics	*Lexis:* words, phrases	*User, conventions:* contexts for the selection of terminology
Quantitative characteristics	*Lexis:* explicit paraphrasing; *Position:* order in enumerations	*User conventions:* terminology, common metaphors
Usage characteristics	*Lexis:* explicit paraphrasing; *Melody:* distinction between input and output; *Pitch:* different voice pitch for warnings	*User, conventions:* terminology
Narrative relations	*Lexis:* explicit paraphrasing, provide means for follow-up questions; *Position:* order	*User:* preferences (follow-up questions or not); *User, conventions:* common metaphors
Conversational relations	*Lexis:* explicit paraphrasing, pre-formulation of answers, etc.	*User, conventions:* preferences, terminology
Organisational relations	*Position:* avoid splitting of groups in speech output, order	*User, conventions:* preferences, terminology

Table 6.2: Presentations in a Natural Language Interface

Part III

A Tool for the Automatic Generation of Human-Computer Interaction in Heterogeneous Environments

In the previous chapters we have developed the ACE framework for an abstract modelling and automatic generation of HC interaction in heterogeneous and dynamic interaction environments. We have also shown how this framework can be put to use.

This part is now dedicated to the description of one possible realisation of the ACE framework. Above all, we want to show the feasibility of the framework. Moreover, the following chapters offer some insights into and guidelines for a practical realisation of the approach—although different approaches can be used to implement the framework.

We have applied the ACE framework to the *Co-Design environment* and will present our results in the following sections. The Co-Design environment is above all developed to realise a parallel and fine-tuned development of the various design aspects of information systems: database structure, views, behaviour and HC interaction. We will concentrate on the HC interaction aspect alone and touch upon other design aspects only when necessary.

Chapter 7 gives an introduction into the Co-Design environment. It describes its aims and principles as well as its architecture and some of its implementational principles and details.

Chapter 8 then illustrates how HC interaction is to be specified in the Co-Design environment. Since the prototype has so far focused on the aspect of application management (system control) and interaction generation, the support offered for the modelling itself is still relatively poor so that we will also point out which improvements of the modelling process are required and planned for the future.

In Chapter 9 we finally present the driver concept that is used in the Co-Design environment to automatically generate HC interaction for various interface system based on a common abstract model. Such drivers have so far been developed for a UNIX command line interface and a Tcl/Tk interface.

Chapter 7

The Co-Design Environment

The following chapter presents a first prototype of the Co-Design environment that is currently being implemented. The Co-Design environment is a *design and execution* environment that enables an integrated specification of the different aspects of information systems: database structure, views, behaviour and HC interaction. Although HC interaction is not its main focus, we have found the prototype convenient to implement and test our previously presented conceptualisations concerning abstract HC interaction modelling and its automatic generation.

This chapter contains an outline of the Co-Design philosophy (Section 7.1) that has led to the development of the design and execution environment. Furthermore, we describe the architecture of the prototype (Section 7.2) and the mechanisms used for its concrete realisation (Section 7.3).

7.1 The Philosophy of the Co-Design Environment

The Co-Design environment is based on the Co-Design methodology for an integrated design and implementation of information systems [Cla00]. The Co-Design methodology was motivated by practical experiences with various projects on application development.

Problems in system development. Projects in our research group include, e.g., the development of several city information systems, an environmental database and application system for the support of researchers and an information system on local research and development activities. These projects have shown that

- the *lack of a common model for design aspects* like database and interaction makes it difficult to communicate and incorporate modifications from one design aspect to other aspects,
- *inconsistencies* arising from modifications are hard to localise and can often only be inferred from a misbehaviour of the application system,
- existing tools usually support the development of applications for a restricted number of target platforms only and hardly consider to address different *interaction paradigms* like graphical interfaces, natural language interfaces, command line interfaces, etc.,
- *style guides* for the application of ergonomic rules, the introduction of corporate identities, etc., are usually not automatically applied in interaction development.

The above collection contains the most prominent problems that our developers had to deal with and that caused significant frustration. These problems are symptoms of lacking support for high requirements in terms of productivity as well as quality of system development. Especially complex application systems pose considerable demands on their developers and render any maintenance and extension extremely difficult [CLS99a].

Basically, the problems in application development can be divided into two problem areas:

Missing integration: A separate design of the individual design aspects suffers from several gaps between specification languages [CT97] and can possibly lead to non-optimal, inconsistent and incomplete designs.

Missing flexibility: A restriction of target platforms that are supported by individual development tools, the requirement to manually enforce certain style guides, etc., limit the accessibility of application systems to an as large as possible number of customers.

The Co-Design methodology, therefore, aims at *(a)* enabling an integrated and well-based development of different design aspects that allows a fine-tuning and a mutual exploitation so that consistency and completeness of design are supported, and *(b)* abstracting from specific applications, execution platforms and target interfaces, thus supplying a high degree of flexibility and portability in system design.

Requirements of design methodologies. The core of the Co-Design methodology is an integrating specification language. When creating such an integrating specification language we have to deal with some antagonistic requirements as the opposite design foci—application and platform—suggest [CLS99b]:

- To allow wide portability, particularly with regard to various interface systems, we have to fine-tune the expressive power of the specification language so that we can take advantage of as many concrete target system capabilities as possible.
- To improve developer productivity, we should include as many specialised application constructs as possible.
- The language has, however, to be kept simple enough to be easy to understand, to use and to implement.

Co-Design. The Co-Design methodology offers a common model for data structures, computational processes and HC interaction. This model is based on a bipartite hyper-graph model [CLS99b]. The key to meeting the above requirements is to distinguish different types of developers (application designers, driver programmers, self-customising end-users, etc.). For each of these developer groups, Co-Design offers *graduated semantics* of the common specification language. This means that the different groups of people creating and customising an application system should use the same system design and only syntactically and semantically different *views* to look at it and manipulate it [CT97]. The abstract HC interaction modelling language introduced in Section 4 is an example of such a 'language view'.

The Co-Design specification language offers several aspects, e.g., branching semantics, where semantics are not fixed and the possibility to program these aspects ensures the required flexibility. A concept for language hierarchies that restricts the amount of manipulation allowed has also been developed [CL99]. A thourough discussion of the Co-Design methodology is provided by Clauß [Cla00].

7.2 Architecture of the Co-Design Environment

With the Co-Design environment prototype we are currently implementing a tool for an integrated specification of information systems and for their direct execution. The architecture of the prototype is shown in Figure 7.1.

Application specification. The application specification contains a description of the application system semantics: of data structures, view definitions, computational and interaction processes, connections between data and

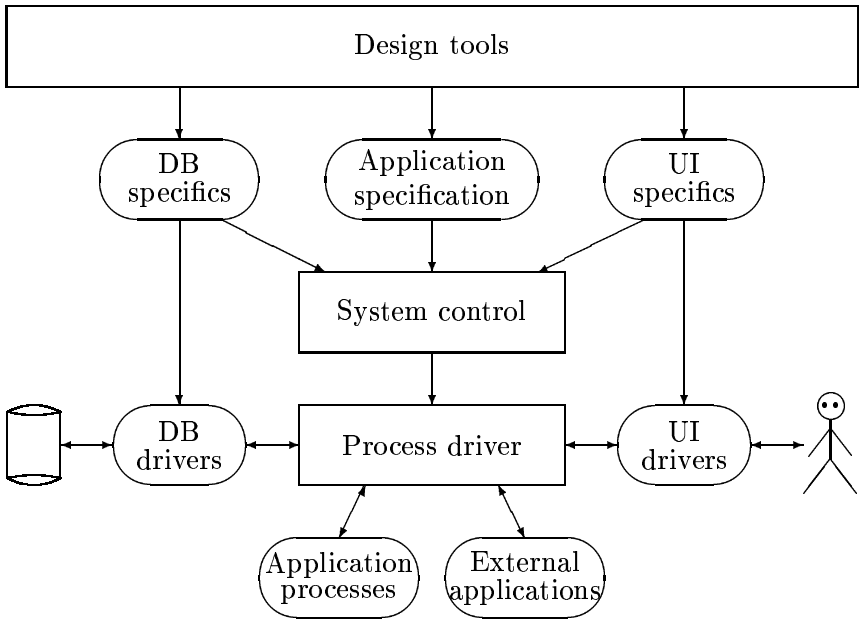


Figure 7.1: The Architecture of the Co-Design Environment

processes, role models, environment models, etc. The application specification is both abstract and platform-independent, i.e., especially the description of HC interaction processes is based on the abstract modelling language introduced in Chapter 4.

DB/UI specifics. In order to enable the incorporation of different platforms—both with respect to database management systems (DBMS) and user interface management systems (UIMS)—the architecture incorporates information on specifics of various DBMS and UIMS.

In terms of UIMS, these specifics contain the rules mapping between abstract HC interaction modelling constructs and concrete constructs available in the interface. Various environment models can also be contained. For the database side, specifics include mappings between data types and database constructs, optimisation rules, etc.

Design tools. Design tools—although they are not yet implemented in the current version of the prototype—are intended to support designers in the development of the application specification and in the description of database and user interface specifics.

Section 8.3 presents some requirements for such a tool support.

System control. The core of the prototype is the system control. The system control reads the application specification and continually checks for the executability of processes. If processes are found executable, the system control starts their execution and moreover performs the scheduling.

Process driver. The process execution lies then in the responsibility of the process driver. Processes are either computational processes or interaction processes.

Computational processes are carried out by the execution of some attached programs or by calls to external programs and applications (which allows to source out some of the system functionality). Interaction processes, which in the Co-Design architecture can concern both interaction with the underlying database and with human users, are carried out by a conversion of the participating data (data to be stored, to be queried, to be displayed, to be elicited) into a form suitable for physical presentation.

DB/UI drivers. The conversion of interaction processes for specific data storage systems and interface systems is done by drivers. We distinguish between virtual drivers and low-level drivers.

Virtual drivers are responsible for the stepwise transformation of data into a form that is fit for the specific target system. This transformation is controlled by the above mentioned DB and UI specifics. Low-level drivers then perform the actual low-level communication with the target system, i.e., they invoke system calls to actually store and retrieve data and to physically present information to the user.

7.3 Realisation of the Co-Design Environment

The current prototype of the Co-Design environment is implemented in Standard C++ for Sun Solaris[1]. For the lexical and syntactical analysis of relevant data, the UNIX tools *flex* and *bison* where chosen.

All specifications, i.e., the application specification (including process descriptions) and DB/UI specifics, are described by means of grammars. The form of these grammars is described in Section 8.1.

For the process execution the concept of term rewriting has been implemented. This means that the process driver accepts as input the respective input data as well as a term rewriting grammar (from the process specification) and performs the specified transformations. This is true for both (internal) computational

[1]Compilers for C++ are, however, available for most platforms. A compilation of the system on Linux has already been successfully tested.

processes and virtual drivers. In the first case, the result of term-rewriting is another data value that can be used in other processes; in the latter case, it is data that is used as input to another virtual driver or to the final low-level driver.

Term rewriting. The mechanism of term rewriting was chosen because it is a relatively simple, yet Turing complete computational model. Integrity constraints are easily transformed into rules and it is reasonably close to algebraic type specifications [CLS99b].

Practice shows, however, that term rewriting is cumbersome and much too slow for complex applications. Functionality that seems trivial (such as the copying of data values) is difficult to realise in the current implementation of the process driver; moreover, it offers only a limited depth for recursions and shows inefficiencies in speed. Using term rewriting alone thus has been proven to be a relatively serious impediment to high performance. This was one of the reasons why calls to external programs and applications have finally been introduced in order to increase the current processing speed.

Team work within the Co-Design environment. The aspect of HC interaction development and generation is only one of various aspects dealt with in the Co-Design environment. Many central components of the environment, above all the system control, are relied upon by different development teams. The modification of such central components is, therefore, discussed by the entire team and an agenda of planned extensions and modifications is maintained.

The development of the overall prototype always has priority over the development of individual aspects.

Chapter 8

Specification of Human-Computer Interaction in the Co-Design Environment

In Chapter 7 we presented an introduction to the current prototype of the Co-Design environment. We have used this prototype to implement and test our ideas and conceptualisations concerning an abstract modelling and automatic generation of HC interaction that have been introduced earlier in this work.

The following chapter puts forth the means and methods that the Co-Design prototype provides for an abstract modelling of HC interaction. Chapter 9 then describes the mechanisms used for automatic interaction generation based on an abstract HC interaction model.

8.1 Specification Language

Originally, the Co-Design prototype provided a relatively simple format for the definition of so-called *control grammars*. A control grammar would include the specification of data, processes and the connections between them, thus describing the structure and application logics of an application system.

By means of such a grammar, any application system is specified by a number of definitions of computational processes (key word PROC), HC interaction processes (UPROC), DB interaction processes (DPROC), data (DATA) and variable instantiations (VAR):

```
⟨appl_system⟩  := PROG (⟨def⟩)
⟨def⟩          := ⟨definition⟩
                  | ⟨definition⟩ , ⟨def⟩
```

Process definitions contain *(a)* an identifier, *(b)* a parameter that either specifies the term rewriting grammar to be used (for computational processes) or process properties that guide the eventual translation into concrete presentations (for interaction processes) and *(c)* incoming and outgoing data.

```
⟨definition⟩   := PROC (ID, ⟨grammar⟩,
                    ⟨input⟩, ⟨output⟩)
                    | UPROC (ID, ⟨properties⟩,
                    ⟨input⟩, ⟨output⟩)
                    | DPROC (ID, ⟨properties⟩,
                    ⟨input⟩, ⟨output⟩)
```

Data definitions contain *(a)* an identifier, *(b)* a data type and *(c)* producing and consuming processes.

Variable instantiations contain *(a)* a data identifier and *(b)* a value.

```
                    | DATA (ID, ⟨datatype⟩,
                    ⟨input⟩, ⟨output⟩)
                    | VAR (ID, ⟨value⟩)
                ...
```

Process properties to guide the translation of HC interaction processes into concrete presentations had so far been left unspecified. Instead default translations based on the types of the participating data were used.

This original specification language is now partially replaced by the abstract modelling language introduced in Section 4. The complete syntax is described in Appendix B. This language substitutes the UPROC and the DATA parts of the original specification language.

8.2 Specification Process

The process of application system specification includes the specification of the overall application structure by means of the definition of data, processes and connections between them as well as the definition of specifics of database systems and interface systems to be addressed.

In the current prototype of the Co-Design environment, specification is still a thouroughly manual process which involves the specification of grammars in formats such as the one presented in Section 8.1 and Appendix B. The process of specification has recently been improved by the introduction of *macros* [See99].

The macro mechanism. Macros can be used to specify complex processes and data elements. They enable a facilitation of design by

- enabling design on different abstraction levels,
- removing redundancies from the design,
- providing the possibility to define macro libraries,
- introducing a high degree of flexibility by allowing different static instantiations as well as a dynamic parametrisation and
- providing the possibility to encapsulate application-irrelevant mechanisms such as the selection of appropriate interface drivers, the incorporation of environment models, etc.

Macros are specified by a *macro definition* that describes whether it is a process macro or a data macro, the input and output of macros as well as parameters available. The incorporation of macros into an application specification is performed by means of a *macro instantiation* where input, output and parameters are linked to elements of the entire application specification.

Macros in the Co-Design environment realise flexibility by the application of various object-oriented techniques such as

- static and dynamic parametric polymorphism (different input/output data and parametric instantiation),
- inclusion polymorphism and to some degree supertype polymorphism (in case of structural supertypes),
- overloading that is decided by the checking of the number and the type of parameters and
- enforced type conversion by the encapsulation of converter processes.

The macro mechanism is presented in detail in the work of Seelig [See99].

8.3 Tool Support

As mentioned before, the current version of the Co-Design prototype has not yet implemented any design supporting tools. This is mainly due to the intention to first concentrate on the provision of stable core functionality and only then to supply additional support.

The lack of design support has, however, led to some frustration among the programmers involved as the writing of control grammars and term rewriting grammars even for testing consumes considerable time, is prone to many programming bugs and makes it difficult for new members of the project team to become acquainted with the system and its methodologies. The implementation of design tools has, therefore, become one of the first priorities for further extensions of the current prototype.

In the following, we would like to list some of the functionality that is planned to be incorporated and is intended to facilitate the design process:

- provision of a graphical design interface to allow for system specification in a graphical notation (such as the one used in Chapter 4) for the overall application structure (data, processes, connections between them) and HC interaction modelling
- translation of the graphical notation into a control grammar including HC interaction specifics
- consistency and completeness checks
  - match of macro definitions and macro instantiations
  - match of characteristics and relations with data and processes already specified
  - check for dead-ends in connections between data and processes
- design suggestions based on heuristics and neighbouring design aspects (views, behaviour)

Chapter 9

The Co-Design Driver Concept for Automatic Human-Computer Interaction Generation

In the Co-Design environment, a driver mechanism is used for the eventual translation of an abstract HC interaction model into concrete physical presentations as well as the actual running of these interfaces. The following chapter first describes the principles behind this driver mechanism (Section 9.1) and then its current realisation (Section 9.2). The current realisation includes the generation of UNIX command line interfaces and Tcl/Tk interfaces based on one common abstract HC interaction model.

9.1 Principles of the Driver Concept

In the Co-Design environment the translation of abstract HC interaction models into concrete presentations and the running of interfaces are performed by drivers. In the following we refer to the concept of interaction objects that was introduced in Section 6.2. Drivers are responsible for the physical presentation of interaction objects. We describe the tasks and the structure of interface drivers as well as the establishment of driver hierarchies.

The generation of interaction objects. The generation of interaction object, i.e., the decision which interaction process to invoke and which data, characteristics and relations to select as relevant, lies in the responsibility of the system control. Since the system control is one of the central components of the Co-Design environment with importance to other development teams too, the modifications required to achieve a customisation of interaction objects have not yet been implemented.

Currently, the system control performs the invocation of processes in the following way: Processes are considered executable if values for all their input data are available. Processes can be either marked as 'automatic' processes or as 'manual' processes. In the first case, they are started automatically in a random fashion. In the second case, they are started on the user's initiative (i.e., the user issuing the respective command). In both cases, the generation of interaction objects then involves the incorporation of *all* the specified data descriptions and associated characteristics and relations.

The presentation of interaction objects by drivers. The physical presentation of interaction objects is then the task of interface drivers which are started on the invocation of the respective interaction process. Interface drivers are required to perform the following tasks:

- They provide the user with the available input data.
- They provide means to the user to insert the required output data.

Both tasks have to take into account the capabilities of the media paradigm addressed in general (e.g., graphical interfaces) and of the interface system in particular (e.g., Tcl/Tk) as well as preferences and requirements of the current user and stylistic guidelines.

Driver structure. All drivers have the structure shown in Figure 9.1. The incoming parameters are information on the interaction environment and the interaction object generated which consists of the type and the value of the input data, the type of the output data and the relevant characteristics and relations. The outgoing parameter of a driver is the output value, i.e., the information extracted from the user.

Internally, drivers communicate with a *receiver*, i.e., they present the transformed input to the receiver and/or extract the required output from the receiver. This receiver can be either a human user or another driver as will be explained below in the context of driver hierarchies. In order to perform this communication, drivers apply rules or programs to convert the input into a form that can be accepted by the receiver and/or they wait for a response and translate that response back into a form that will be used for further processing by other processes. Examples of such transformation rules as they are used in the current prototype of the Co-Design environment will be given in Section 9.2.

Virtual vs. low-level drivers (Driver hierarchies). The distinction between virtual and low-level drivers has already been introduced in Section 7.2.

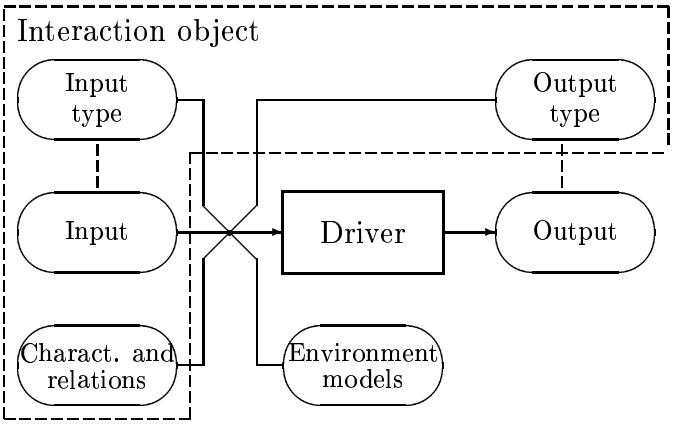


Figure 9.1: Driver Structure

Virtual drivers perform a stepwise transformation of abstract into concrete constructs; low-level drivers perform the actual and direct communication with the specific target interface. Whereas low-level drivers are thouroughly target-dependent, virtual drivers will usually show less (possibly close to none) focus on specific interface systems so that in principle they could be employed for the addressing of distinct (though similar) interface types such as for the translation of abstract models into representations fit for graphical interface systems in general.

Consequently, drivers can be organised into driver hierarchies. A translation process that traverses such a hierarchy will perform a stepwise concretisation of the current interaction object until it finally reaches the final low-level translation. Figure 9.2 shows an example of a possible driver hierarchy.

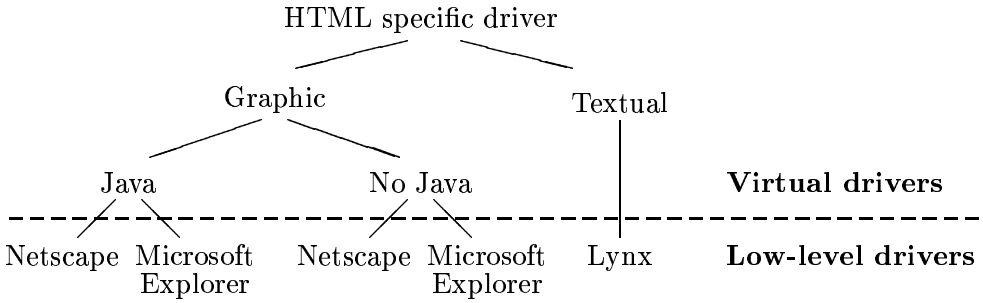


Figure 9.2: A Virtual Driver Hierarchy

In the driver hierarchy shown several drivers are organised that perform a translation into HTML customised for different browser types. Please note that a distinction between low-level drivers for different systems like Netscape and Microsoft Explorer is not actually necessary. In cases where the differences between

target interfaces are only minimal, they can be accommodated by one driver that internally realises this distinction. This also helps to reduce the width of hierarchies and the efforts to be put in driver development.

For the process of translation, virtual and low-level drivers are nested so that output of a more high-level driver serves as input to a less high-level driver and output of a less high-level driver serves as input to a more high-level driver. Environment models are input to drivers in all hierarchy levels. The nesting of interface drivers is illustrated in Figure 9.3.

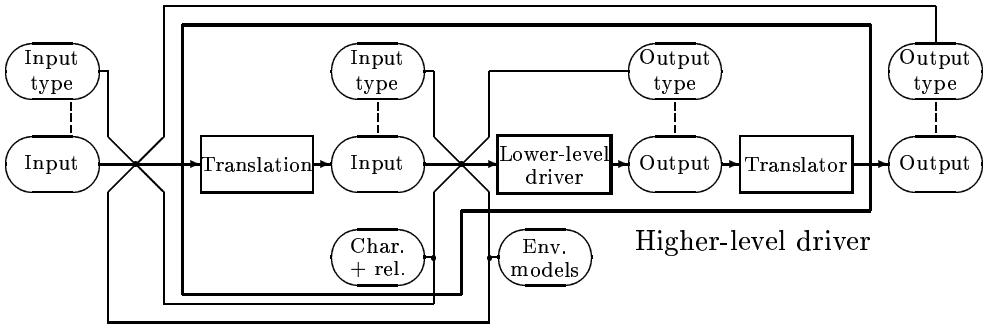


Figure 9.3: Nesting Interface Drivers

9.2 Current Realisation of the Driver Concept

The current version of the driver mechanism offers the following features:

- a structure-based translation of an abstract HC interaction model for a UNIX command line interface [See98] and a Tcl/Tk interface [Rad99] and
- a translation of selected semantical concepts for abstract HC interaction modelling influenced by selected properties of the interaction environment for a Tcl/Tk interface [Rad99, Rad00].

9.2.1 A UNIX Command Line Interface

The structure-based translation was first implemented in a very simple version for a UNIX command line interface. A UNIX command line interface is generated in two steps:

1. A virtual driver generates
   - a partial program tree for displaying information based on the input data of the driver,

- a partial program tree for reading information from the user based on the output type of the driver and
- a merged program tree consisting of the combined display and read tree.

The types supported are the **1** type (void), bool, natural numbers and lists of these types.

2. The low-level driver interprets the merged program tree and thus realises the actual communication.

Following are a few examples of translation rules:

Example 9.1: Displaying bool

```
true() => prtstr(yes)
false() => prtstr(no)
```

■

The type bool is displayed by printing corresponding strings. These strings can, of course, be replaced by variables taken from a style file which can be set to different languages.

Example 9.2: Reading lists

```
list(@x) => cli1(case(read_alternative(continue,stop),
                      return(cons(call(2),call(1))),
                      return(nil))
                 @x)
```

■

Lists are read from the user element by element; after each element the user can choose to continue or to stop. By the above rule a tree (cli1) with two branches (case and @x) is defined. Execution enters the first branch and reads the user's selection. In the stop case, an empty list (nil) will be returned. In the continue case, the return is a concatenation of the second branch's return (reading an element of the respective list type) and the return of the repeated process.

Display and read trees are merged either by a simple concatenation or by a more complex merge where display information is inserted into the read tree itself. An example for this is the reading of a list of numbers with a repeated prompt (@x):

Example 9.3: Merging display and read trees

```
merge(@x,cli1(@y,..)) => cli1(seq(@x,@y),..),
                 if @x=seq(..) or prtstr(..) or prtnat(..) or newline()
```

■

A possible program tree in a more easily readable graphical notation is shown in Figure 9.4. The low-level driver traverses the tree and executes its nodes: it displays information on meeting a prtstr node, it changes to another program branch on meeting a call node, etc.

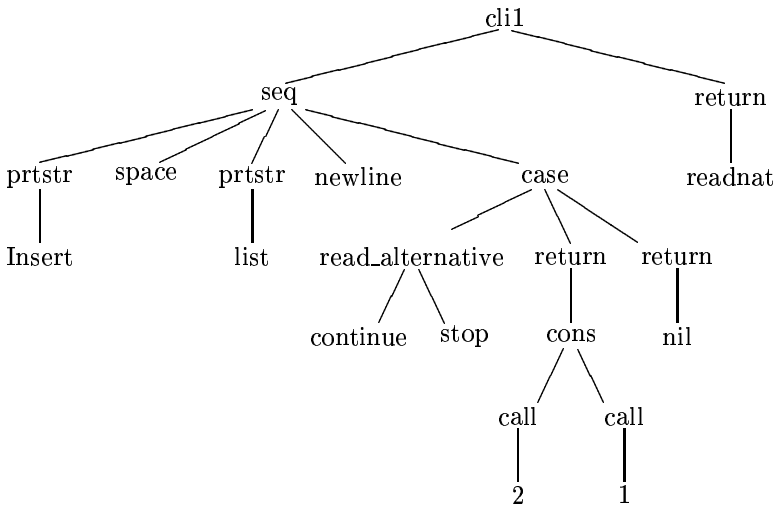


Figure 9.4: A Program Tree for a UNIX Interface

9.2.2 A Tcl/Tk Interface

In the case of the Tcl/Tk interface, the presentation of interaction objects employs similar principles to those used by the purely structure-based translation for a UNIX command line interface. Due to the higher complexity of the semantics-based translation, some extensions and modifications have been introduced. Above all, the process of giving a physical presentation to an interaction object is split into two steps as is shown in Figure 9.5.

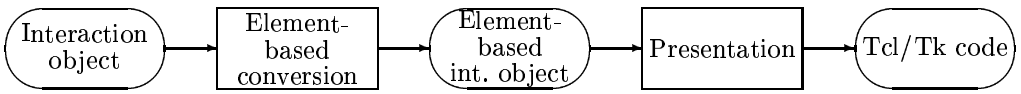


Figure 9.5: Presenting Interaction Objects for a Tcl/Tk Interface

In the first step, an element-based representation of the interaction object is created, i.e., based on the specified *data relations* individual data elements are combined into complex elements that loosely correspond to typical constructs of graphical interface systems. In the second step, the element-based interaction

object is transformed into Tcl/Tk code, i.e., complex data elements are translated into Tcl/Tk interface constructs and construct settings are chosen according to the specified *data characteristics*.

These translations have been conceptually and practically substantiated, i.e., the intermediate language for the description of element-based interaction objects has been developed (cf. Appendix C) and rules for translation have been laid down and partially implemented. The Tcl/Tk driver is continuously improved in order to render the translation yet more flexible and to incorporate further rules and semantical concepts into the translation.

Element-based interaction objects. Element-based interaction objects consist of

- a number of interaction *elements* that each combine a PROVIDE part (data to be provided but not shown directly), an EXTRACT part (data to be extracted from the user) and a SHOW part (data to be directly shown to the user) some of which can be void,
- *groups* of such elements,
- for each interaction element or element group *properties* such as relevance and emphasis and
- the assignment of *values* to data to be shown or provided and simple variable names to data to be extracted, respectively.

Interaction elements in such element-based interaction objects are already oriented at constructs which are typically available in graphical interface systems. In principle they can also be used for translations into graphical interface systems other than Tcl/Tk.

Generating element-based interaction objects. For the generation of element-based interaction objects out of original interaction objects the following rules are currently used:

Rule 1: If in the original interaction object output data exists without a (conversational) relation to input data,
then it is mapped onto the EXTRACT part of an interaction element while the PROVIDE and SHOW parts remain void.

Rule 2: If in the original interaction object input data exists without a relation to other (input or output) data,
then it is mapped onto the SHOW part of an interaction element while the PROVIDE and EXTRACT parts remain void.

Rule 3: If in the original interaction object output data A exists with only a default relation to input data B,
then A is mapped to the EXTRACT part of an interaction object while B is mapped to the SHOW part and the PROVIDE part remains void.

Rule 4: If in the original interaction object output data A exists with only a suggestion relation or a choice relation to input data B,
then A is mapped to the EXTRACT part of an interaction object while B is mapped to the PROVIDE part and the SHOW part remains void.

Rule 5: If in the original interaction object output data A exists with a default relation to input data B and a suggestion relation or a choice relation to input data C,
then A is mapped to the EXTRACT part of an interaction object while B is mapped to the SHOW part and C is mapped to the PROVIDE part.

Rule 6: If in the original interaction object data is related by an example relation, a summary relation or a group relation,
then a group of the respective interaction elements is created.

Rule 7: Additional interaction elements are created for each output data which serve as labels[1] to the respective interaction elements. They are chosen based on the categorisation (qualitative characteristic) of the data (In this case, the stylistic model needs to provide appropriate values.) or based on the data type if a categorisation does not exist (In that case, simple labels like 'Name' or 'Number' are created.). Moreover, a label and the respective interaction element are combined into a group.

Emphasis and relevance characteristics (which can be different for different users) are simply transferred from the original interaction object. Concrete data values for input data are retained in the VALUES component which also contains variable names for the expected output values which are generated in order to enable the assignment of the extracted values to the respective output data. Values for interaction elements that have been created as labels are selected from a choice of textual labels in different languages and iconic labels. For the Tcl/Tk driver, we incorporate this choice into the stylistic model as shown in the following example:

[1]Since most graphical widgets in Tcl/Tk cannot directly be associated with a label we have chosen this 'work-around' to tag labels to individual elements.

Example 9.4: Labels in the stylistic model

```
LABELS (hotel: ((english, 'Hotel'),
                (french, 'Hotel'),
                (german, 'Hotel'),
                (iconic, $hotel.jgp)),
        arrival: ((english, 'Day of arrival'),
                  (french, 'Jour d'arrivée'),
                  (german, 'Anreisetag'),
                  (iconic, $arrival.jgp)),
        ...)
```

■

Generating Tcl/Tk code. For the translation of element-based interaction objects into Tcl/Tk code the following rules have been developed:

Rule 8: Each interaction element with a filled SHOW part and void PROVIDE and EXTRACT parts is mapped onto a simple text box.

Rule 9: Each interaction element with a filled EXTRACT part and void PROVIDE and SHOW parts is mapped into simple entry fields.

Rule 10: Each interaction element with filled PROVIDE and EXTRACT parts and void SHOW parts is mapped onto a combo box[2] without a displayed default value where clicking on the pull-down button displays the PROVIDE part.

Rule 11: Each interaction element with filled PROVIDE, SHOW and EXTRACT parts is mapped onto a combo box with the SHOW part being displayed as a default value and where clicking on the pull-down button displays the PROVIDE part.

Rule 12: For the arrangement of elements, elements of high priority or emphasis are displayed first and grouped elements are displayed next to each other.

The above rules are *examples* of possible translations. By the use of another rule set, very different effects on the actually created presentations of interaction objects can be achieved. Even this set of relatively simple rules shows, however, how a number of semantical constructs used for the abstract modelling of interaction can be exploited for an automatic generation of HC interaction.

[2] A combo box construct does not exist in Tcl/Tk; it was simulated by the combination of several other interface constructs.

- For the generation of element-based interaction objects, we have exploited conversational data relations (default, suggestion, choice) as well as qualitative characteristics (application-dependent categorisations).
- For the translation into Tcl/Tk code, we have exploited groupings and quantitative data characteristics (emphasis, relevance).
- A customisation is possible by means of the arrangement of interaction elements based on different emphasis and relevance values and by the choice between different labels depending on the user's preferences.

The application of each rule requires to match the specified pattern either in the original interaction object (for the first step of generation) or in the element-based interaction object (for the second step of generation). Currently, this pattern matching involves an often complex traversing of the input tree to the driver. As long as the current mechanisms for this pattern matching are sufficient, the incorporation of new or modified rules is, however, easily possible.

Figures 9.6 and 9.7 show two examples of Tcl/Tk interface as they could be created by the Tcl/Tk driver. The interface shown in Figure 9.6 depicts an interface for an English speaking user with 'normal' emphasis values; Figure 9.7 depicts an interface for a German speaking user with a high emphasis value set on the price category.

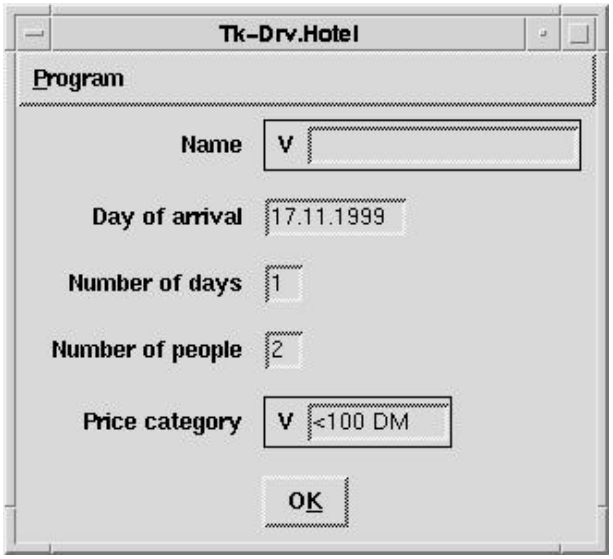


Figure 9.6: A Tcl/Tk Interface Created by the Co-Design Prototype (1)

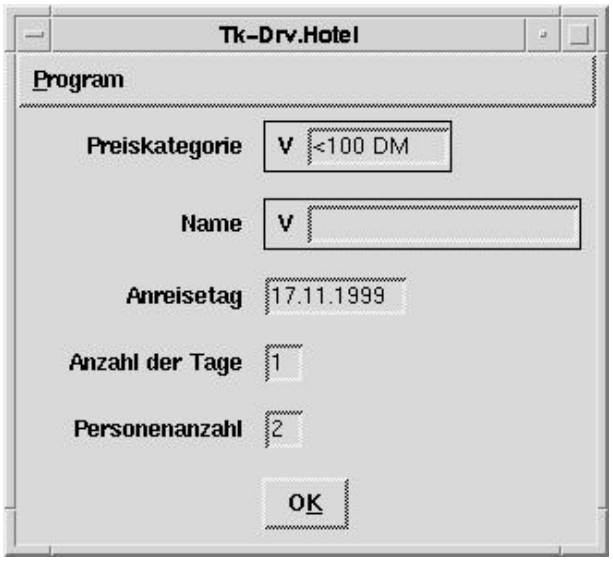


Figure 9.7: A Tcl/Tk Interface Created by the Co-Design Prototype (2)

Conclusion and Outlook

> 'GPP Feature?' said Arthur. 'What's that?'
> 'Oh, it says *Genuine People Personalities*.'
> 'Oh,' said Arthur, 'sounds ghastly.'
> A voice behind them said, 'It is.' The voice was low and hopeless and accompanied by a slight clanking sound. They spun around and saw an abject steel man standing hunched in the doorway.
> 'What?' they said.
> 'Ghastly,' continued Marvin, 'it all is. Absolutely ghastly. Just don't even talk about it. Look at this door,' he said stepping through it. The irony circuits cut into his voice modulator as he mimicked the style of the sales brochure. '*All the doors in this spaceship have a cheerful and sunny disposition. It is their pleasure to open for you, and their satisfaction to close again with the knowledge of a job well done.*'
> As the door closed behind them it became apparent that it did indeed have a satisfied sigh-like quality to it. 'Hummmmmmmyummmmmmm ah!' it said...
> 'Come on,' he droned, 'I've been ordered to take you down to the bridge. Here I am, brain the size of a planet and they ask me to take you down to the bridge. Call that *job satisfaction*? 'Cos I don't. ... Thank you, the marketing division of the Sirius Cybernetics Corporation,' said Marvin and trudged desolately up the gleaming curved corridor that stretched out before them. '*Let's build robots with Genuine People Personalities*, they said. So they tried it out with me. I'm a personality prototype. You can tell can't you? ... Here's another of those self-satisfied doors. *Life!* Don't talk to me about life.'
>
> ¿From *The Hitchhiker's Guide To The Galaxy* by Douglas Adams

This work dealt with the development of human-computer (HC) interaction for heterogeneous and dynamic environments. Such environments are often encountered by information systems that have to

- satisfy a variety of *users* with different interests, preferences, abilities, expectations and cultural backgrounds,
- cope with differing *technical equipment*,
- that moreover wish to or have to occasionally change the appearance of interfaces by new *stylistic guidelines* and
- that finally want to take the *interaction history and goal* into account to provide information and offer interaction flows that are relevant and motivating.

Truly heterogeneous and dynamic environments used to be rare but nowadays gain more and more significance due to the increasing penetration of networked

environments (most of all the Internet) into daily life, thus giving access to information systems to an extremely diverse audience. E.g., the audience checking into an online travel agency will be undoubtedly much more diverse than in the most metropolitan of cities.

In order to keep this audience satisfied (and make it want to come back), the audience has to be given what it really wants and needs. Besides high-quality information and sufficient functionality, a key factor in keeping the audience satisfied is *human-computer interaction* which makes all this information and functionality at all available. HC interaction needs to be customised to offer appropriate information, appropriate flow (navigation) and an appropriate form (appearance of interaction) depending on the user's tastes and interests, on whether the information can be displayed on his/her machine, etc.

In previous approaches, only selected aspects of such a customisation (mostly interaction form) have been addressed in the form of some prototypical solutions. What has been missing so far, is a more general and thourough approach.

In this work, we have developed a methodical framework that describes the development of HC interaction for heterogeneous and dynamic environments in a *complete* and *general* fashion. In the following, we will provide a summary of our findings and will moreover point out some possible criticisms and future extensions.

Summary

The ACE framework, which was developed in this work, was created to capture the tasks and their structure that are involved in the development of HC interaction for heterogeneous and dynamic environments. The structure of the framework is shown in Figure 10.1.

The *scope* dimension of the framework consists of the components of interaction: flow, content and form. The *abstractness* dimension describes the process of development from an abstract model of interaction (interaction space), to the evaluation (the selection of relevant parts of the model into interaction objects) and finally to the presentation (giving a physical shape to interaction objects). The process of development is further accompanied by information on the interaction environment which is kept in various environment models.

Principles of the ACE framework. Underlying ACE are the principles of abstraction, completeness and explicitness:

Abstraction refers to the fact that a model of HC interaction is required that should (ideally) pay no attention to possible interaction situations at all. For one thing, there might be far too many situations; for another thing,

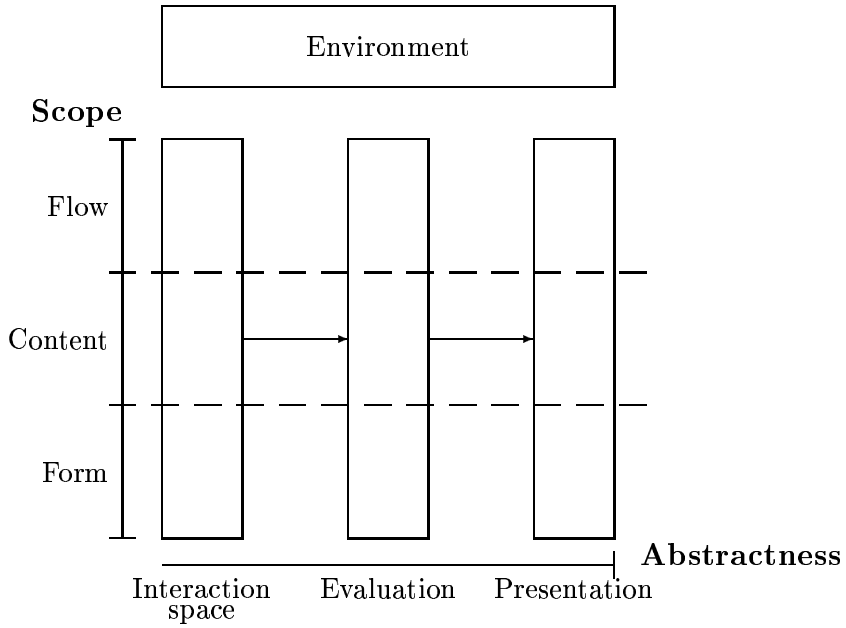


Figure 10.1: The Structure of the ACE Framework

new situations might arise that we had not thought of before and that would make our design obsolete. Instead, the modelling of HC interaction should concentrate on the semantics of interaction alone: on which information could possibly be relevant, what the characteristics of that information are, etc.

For the abstract modelling of HC interaction, we have developed a language (cf. Section 4).

Later on, the abstract model will be concretised according to the abstractness dimension of the framework, e.g., processes to be realised are selected depending on their characteristics and the current interaction situation, information is selected and a physical form is given to all relevant elements so that concrete presentations of interaction are generated on the basis of the abstract model.

Completeness is expressed by the scope dimension of the ACE framework. It simply requires that all the components of HC interaction (as long as they are considered for a customisation) are to become part of the abstract HC interaction model and undergo the process of concretisation. This concerns the flow of interaction, the content of interaction and the form of interaction.

The developed modelling language contains constructs for all these components (cf. Section 4.5).

Explicitness is the call for a direct specification not only of the interaction semantics and the influencing environmental information, but above all of the mechanisms that are applied in order to perform a concretisation of interaction, i.e., they should not be hidden inside complex translations. This makes it possible to relatively easily replace or redefine these mechanisms so that new or changing interaction situations can readily be incorporated.

In the ACE framework, rules are used to translate individual modelling constructs into concrete presentations (cf. Section 6). The translation thus concerns *(a) individual* properties instead of complex application objects, *(b)* properties that capture the *semantics* of interaction and *(c)* constructs which are *universally* used throughout the specification. While some of these aspects have also been realised in other approaches, the *combination* of these features is new. The exchange of translation rules is hence facilitated while consistency of presentations across the application is ensured.

The principles of abstraction, completeness and explicitness ensure that one common HC interaction model can serve as the starting point for interaction generation tailored to a varied audience and that the mechanisms used for this generation are moreover relatively easily interchangeable and do not necessarily require a re-design of the interaction model itself.

Modelling HC interaction within the ACE framework. We have introduced a number of constructs for the abstract modelling of HC interaction. These constructs describe processes relevant for HC interaction, the participating data as well as a number of characteristics and relations that are intended to capture the meaning of processes and data.

It is these characteristics and relations that are a completely new feature of interaction modelling. They enable to specify the meaning of interaction on a level that abstracts from concrete interaction situations (above all interface systems and paradigms) and rather concentrates on questions like: Do these data have a connection? Which sort of a connection? Is this data important? What purpose does this process serve?.

Only later, when interaction is concretised, are these characteristics and relations (along with conditions attached to them) used to decide on the relevance of specific constructs in the encountered interaction situation and on the physical appearance to be given to them.

Generating HC interaction within the ACE framework. The generation of HC interaction based on an abstract model is very much dependent on the specific interface system to be addressed. In this work we have, therefore,

confined ourselves to the provision of a general architecture for interaction generation and to the development of a number of case studies to illustrate possibilities of interaction generation.

The architecture proposed relies on the concept of interaction objects and on two types of rules:

Evaluation rules decide which parts of an abstract model are relevant in a certain interaction situation. They do this either by checking the conditions attached to constructs or by analysing the characteristics and relations themselves based on various heuristics. Relevant parts are then selected into interaction objects.

Presentation rules decide on the physical shape to be given to interaction objects. They realise type mappings and use data characteristics and relations to set properties of the chosen interface constructs and their arrangement.

A prototype. To show the practical impact of the ACE framework, we have used a prototype that is currently being developed in our research group. This prototype is intended to realise an integrated design and implementation of information systems in all their aspects: data structure, views, behaviour and HC interaction.

With regard to the HC interaction component, we have realised the generation of interaction based on a common abstract HC interaction model for a UNIX command line interface and a Tcl/Tk interface.

Criticism and Open Problems

I have personally made the experience that automatic interaction generation in general and the use of an abstract and general interaction model to achieve customisation in particular, raise questions concerning the usefulness and the feasibility of such an approach. The following section is intended to tackle some of the most prominent questions and will furthermore provide an outlook to possible future work.

Do we really need this? Is customisation of interaction really so important to make a model for it necessary?

Customisation is a vital aspect in retaining customer satisfaction and loyalty. In a recent paper, we have used the analogy to the early days of car manufacturing to show the importance of customisation [FL00]: In 1921, the Ford Motor Company had reached a market share of 51% due to the introduction of the black Model T Ford—a car that was extensively produced in a never varying style:

same colour, same equipment. By 1927, however, Ford's market share had fallen back to 11% because their highly specialised production line and structure were unable to respond to new market requirements. Customers had grown tired of the Model T Ford and moved to other, more promising cars [AW74].

The same paper shows that this situation is similar to today's information economy: If providers are not able to meet customer requirements, customers will move to where their requirements and expectations are better catered to. To keep an as large as possible number of customers satisfied, online customisation of information offers is indispensable.

Do we want it? It is often doubted that automatic customisation will actually satisfy users. One issue is lacking quality and appropriateness of customisations. Another issue is that humans tend to refuse the idea of computers reasoning about them and deciding what is best for them.

Undoubtedly, the quality and appropriateness of automatically performed customisation is often not able to meet fine-tuning of interaction that has been performed by human designers. And yet, this will change in the future when the reasoning powers of computers and their ability to take background knowledge and other sources of knowledge into account are improved. A better performance in automatic customisation will also help to overcome users' aversions.

This is not a reason, however, why an automatic generation of customised interaction should not be considered now and why a model for the development of flexible HC interaction could not help to improve the understanding of interaction mechanisms and could not help to develop better ones.

Is all the effort worth it? The development of an abstract HC interaction model along with the mechanisms for its concretisation surely involve more effort than the straightforward development of interfaces does. This effort, however, pays off in the context of heterogeneous and dynamic environments which are exactly the context of this work.

A truly abstract HC interaction model can be used to address as many different interaction situations as is desired without requiring any redesign of the model. The actual customisation is instead performed during the concretisation of this model by evaluation rules and presentation rules. The fact that these rules are laid open and are not hidden in complex translations (or even designers' minds) makes it, however, easy to exchange or modify them in order to address new interaction situations.

Where is my control of the design outcome? The developers of an abstract interaction model have no more control of the actual appearance. Instead, the appearance is decided upon in the presentation phase which is independent

from the abstract model. It is even suggested that presentation rules are developed on the basis of suggestions by interface design experts so that the modelling of interaction and the presentation often lie in the responsibility of completely different people.

This decision has been made for two reasons: firstly, the separation into modelling and concretisation was done to ensure the independence of the model from specific interaction situations, thus keeping the model prepared for being used in different interaction situations; secondly, this distinction also allows designers to concentrate on what they are experts in: application designers on the application, interface designers on the interface.

A tight connection between both processes of development is, however, required. Application designers and interface designers have to communicate extensively on the meaning of the constructs employed for HC interaction modelling, just as interface designers and the implementers of rules have to communicate on the realisation of interface guidelines.

What else? A number of future extensions of this work are conceivable. In the following we mention only two:

Facilitation of the development process: For the development of the abstract HC interaction model, of the environment models and of the rules for evaluation and presentation, design support should be offered.

This includes the possibility to use a design tool that frees the designer from modelling formats such as the one which has been presented in this work. Graphical notations are often considered to be more easily usable. A translation into the actual modelling format also needs to be provided.

Furthermore, the developed models could be checked for consistency and heuristics for their completeness could be applied.

Another aspect is the support of the development process. Designers could be guided according to various design strategies, design steps could be suggested, etc.

Exploitation of neighbouring design aspects: HC interaction development is tightly connected to the development of database views (interaction content) and the overall behaviour of the application system (interaction flow).

A theoretical model for these dependences exists but so far no practical realisations have been developed. Mechanisms that exploit these three interrelated design aspects would be helpful for the design of all three aspects.

So what? The contribution of this work is the provision of a methodical framework that describes the development of HC interaction for information systems in heterogeneous and dynamic environments in a thourough and general fashion. We have pointed out that justified criticisms can be given and that several extensions are still possible and required in order to render this framework usable for a wider audience. The development of new computing paradigms, reasoning paradigms, etc., might even lead to a completely different approach to HC interaction modelling and customisation.

So what will remain of the ACE framework?

Of course, we can hardly reason on future discoveries of new computing paradigms and the like. Given the current state of the art, we can, however, state the following: The methodical approach which the ACE framework represents unifies a number of concepts some of which are new and which in their entirety constitute an approach that is more capable than existing ones.

What is above all new is the use of abstract semantical concepts for the modelling of HC interaction. Moreover, the modelling of HC interaction also includes an abstract and semantics-based specification of the flow and the content of interaction, not only of interaction form.

Thus, the ACE framework achieves two things which set it apart from other approaches:

- The use of abstract semantical concepts ensures the independence of the HC interaction model from its concrete realisation, i.e., of the conceptual model from the physical model—an aspect which is state of the art in other areas of computer science but has so far to some extent been neglected in the field of HC interaction. Thus, a change of the circumstances of a concrete realisation (i.e., changes in the interaction environment) will not necessarily require changes in the HC interaction model. HC interaction development is rendered more productive and efficient.

- This is also true for the components of interaction flow and interaction content. These components are separated from the overall application functionality as far as possible so that their customisation is equally efficient.

The successfulness of customisable HC interaction lies in the ability to provide a level of abstraction sufficient enough to cope with a wide range of tasks, interaction partners and interaction media. Though the call for a completely abstract interaction model might not be justified in all cases of applications, the idea of abstractness and completeness contributes to a large extent to the possible degree of flexibility and customisability that can be achieved.

The ACE framework will hopefully be a starting point and an inspiration for further research reaching out for the target of highly customisable HC interaction.

Appendix A

A Model for the Dependences between Design Aspects

A.1 Basic Notions of Category Theory

In the following, we will explain some of the basic notions of category theory [AL91, Poi92, Cam99] as required to outline the model in Section A.2.

Definition 9 *A* category $\mathcal{C}$ *consists of*

- *a class O of objects,*
- *a class M of morphisms (or arrows),*
- *a pair of functions, dom (domain) and cod (codomain), from M to O— describing the source and the target of morphisms,*
- *for each $x \in O$, an identity morphism I_x and*
- *a partial operation of composition on M, the composition of f and g (if it exists) being written fg.*

A category satisfies the following axioms:

- The composition fg exists if and only if $cod(f) = dom(g)$. If this holds then $dom(fg) = dom(f)$ and $cod(fg) = cod(g)$.
- If fg and gh are both defined, then $(fg)h = f(gh)$ (associativity).
- $dom(I_x) = cod(I_x) = x$.
- If $dom(f) = x$ and $cod(f) = y$, then $I_x f = f = f I_y$.

Imagine, e.g., the category $\mathcal{SET}$ of sets. In this category, sets are the objects and functions (like union, etc.) are the morphisms. The identity morphism maps a set onto itself.

Definition 10 *Morphisms between categories are called* functors.
Let $\mathcal{C} = (O, M)$ and $\mathcal{C}' = (O', M')$ be categories.
A functor F consists of a pair of mappings from O to O' and from M to M'. It satisfies the following conditions:

- $(fg)F = (fF)(gF)$ *whenever fg is defined.*
- $I_xF = I_{xF}$ *for all $x \in O$.*

We can, e.g., define the functor $\mathcal{SET}2\mathcal{BAG}$ that maps the category $\mathcal{SET}$ to the category $\mathcal{BAG}$, the category of bags (multisets). The mapping of objects is straightforward: during the transformation into a bag the instance of any set is kept. The mapping of functions drops the checking for multiple elements.

A.2 A Dependency Model

We can identify two dimensions on any level of information system design: locality and dynamics (cf. Figures 1.4 on page 29 and 5.1 on page 111). Hence, we distinguish the following four aspects of system design:

Database structure: global, static.
The database structure consists of the database schema along with database instantiations that are determined by the schema.

Behaviour: global, dynamic.
Behaviour is described by the definition of all possible functionality (whether observable from the outside or not) of the information system.

External views: local, static.
External views are created by the application of view translations to a database instance. Thus, they make the content of the database locally available, depending on the application context.

HC interaction: local, dynamic.
HC interaction is described by a translation of the overall behaviour into behaviour that is locally observable, i.e., it includes only that part of the system behaviour that is involved in the communication with specific human users.

A model that captures the relationships between these four design aspects is described by Clauß et al. [CLT97]. It was developed to exploit analogies between relations in schema and view design on the one hand and relations in behaviour and HC interaction design on the other hand. The model relates the static to the dynamic dimension on the one hand and the global to the local dimension on the other hand. A theoretical substantiation of the model and its consequences is the topic of ongoing research [Cla00].

Static/dynamic. The model considers an information system as a category $\mathcal{IS}$. The class DB of objects of this category is described by the database schema. It determines possible database instances. The class Beh of morphisms describes the processes of the information system that make up the system's behaviour (cf. Figure A.1). A process $f \in Beh$ is either computational or interactive; processes transform one database instance into another. The relation between processes and schema is captured by the *dom* and *cod* functions.

Thus, together with the database instances, processes make up the overall behaviour of the information system so that the category $\mathcal{IS}$ captures the relationship between the static and the dynamic aspect of information systems.

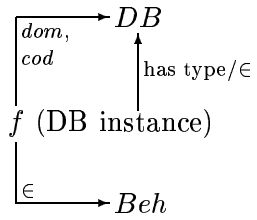


Figure A.1: An Information System as Category

Global/local. Based on the category $\mathcal{IS}$ we can then define a functor F. This functor is responsible for a localisation of the category $\mathcal{IS}$, i.e., the information system behaviour which comprises both the database structure and the transforming processes. The localisation functor F is illustrated in Figure A.2.

F maps both the objects and the morphisms of the category $\mathcal{IS}$ onto objects and morphisms of another category $\mathcal{LIS}$. The object mapping corresponds to *view translations* as illustrated on the right hand side of Figure A.2. View translations transform a database instance into external views and thus make the database content locally available—or rather *observable* (cf. Section 2.2 on interactive computing).

In analogy to view translations, the morphism mapping extends view translations by translations of process instances onto those processes only that are locally

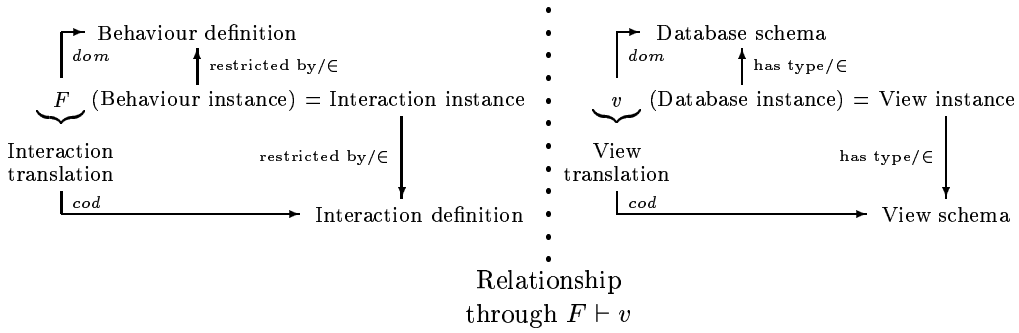


Figure A.2: The Localisation Functor

observable: *interaction translations*. Locally observable processes are interaction processes that are responsible for the provision of information to the user and/or the extraction of information from the user and computational processes that directly support interaction processes by performing required data manipulation.

The category $\mathcal{LIS}$ thus describes HC interaction: information to be exchanged along with interaction processes and supporting processes that realise this exchange. The localisation functor F thus captures the relation between the global and the local dimension of an information system.

Appendix B

An Exemplary Syntax for the Conceptual Modelling of Human-Computer Interaction

```
⟨hci⟩              ::= HCI (DATA (⟨data_list⟩),
                       INTERACTION (⟨interaction_list⟩),
                       COMPUTATION (⟨computation_list⟩),
                       CHARACTERISTICS (⟨charact_list⟩),
                       RELATIONS (⟨relations_list⟩))
⟨data_list⟩        ::= ⟨data⟩
                       | ⟨data⟩, ⟨data_list⟩
⟨interaction_list⟩ ::= ⟨interaction⟩
                       | ⟨interaction⟩, ⟨interaction_list⟩
⟨computation_list⟩ ::= ⟨computation⟩
                       | ⟨computation⟩, ⟨computation_list⟩
⟨charact_list⟩     ::= ⟨characteristic⟩
                       | ⟨characteristic⟩, ⟨charact_list⟩
⟨relations_list⟩   ::= ⟨relation⟩
                       | ⟨relation⟩, ⟨relations_list⟩
```

```
⟨data⟩    ::= ⟨d_id⟩: ⟨name⟩: ⟨d_type⟩
              | NUL
⟨d_id⟩    ::= D#⟨n_list⟩
⟨n_list⟩  ::= ⟨number⟩
              | ⟨number⟩.⟨n_list⟩
⟨name⟩    ::= ⟨word⟩
⟨d_type⟩  ::= ⟨basic⟩
              | ⟨complex⟩
```

```
⟨d_list⟩   ::= ⟨data⟩
             | ⟨data⟩, ⟨d_list⟩
⟨basic⟩    ::= STRING | CHAR | NAT | REAL | BOOL | DATE
⟨complex⟩  ::= SET (⟨data⟩)
             | LIST (⟨data⟩)
             | RECORD (⟨d_list⟩)


⟨interaction⟩  ::= ⟨i_id⟩: IN (⟨cluster_list⟩), OUT (⟨cluster_list⟩),
                   ACT (⟨actors⟩)
⟨i_id⟩         ::= I#⟨number⟩
⟨computation⟩  ::= ⟨c_id⟩: IN (⟨cluster_list⟩), OUT (⟨cluster_list⟩),
                   ACT (⟨actors⟩)
⟨c_id⟩         ::= C#⟨number⟩
⟨cluster_list⟩ ::= CLUSTER (⟨conditions⟩): ⟨comp_list⟩
                 | CLUSTER (⟨conditions⟩): ⟨comp_list⟩, ⟨cluster_list⟩
⟨comp_list⟩    ::= COMP (⟨conditions⟩): ⟨did_list⟩
                 | COMP (⟨conditions⟩): ⟨did_list⟩, ⟨comp_list⟩
⟨did_list⟩     ::= ⟨d_id⟩
                 | ⟨d_id⟩, ⟨did_list⟩
⟨actors⟩       ::= ⟨a_list⟩ ? ⟨conditions⟩
                 | ⟨a_list⟩ ? ⟨conditions⟩, actors
⟨a_list⟩       ::= ⟨a_id⟩
                 | ⟨a_id⟩, ⟨a_list⟩
⟨a_id⟩         ::= A#⟨number⟩


⟨characteristic⟩ ::= ⟨dir_char⟩
                    | ⟨indir_char⟩
                    | ⟨p_char⟩
⟨dir_char⟩       ::= ⟨ch_id⟩: ⟨d_id⟩, ⟨dch_type⟩, ⟨quantity⟩,
                     ⟨context⟩, ⟨conditions⟩
⟨indir_char⟩     ::= ⟨ch_id⟩: ⟨d_id⟩, ⟨ich_type⟩, ⟨quantity⟩,
                     ⟨context⟩, ⟨conditions⟩,
                     ⟨priority⟩, ⟨visibility⟩
⟨ch_id⟩          ::= CH#⟨number⟩
⟨dch_type⟩       ::= MANIP
                   | SELECT
                   | RELEV
                   | EMPH
⟨ich_type⟩       ::= ALERT
                   | APPL:⟨word⟩
```

```
⟨quantity⟩        ::= ⟨ch_equation⟩
                    | ⟨ch_range⟩
                    | ⟨ch_comparison⟩
⟨ch_equation⟩     ::= ⟨che_value⟩
                    | (⟨ch_equation⟩ ⟨f_op⟩ ⟨ch_equation⟩)
⟨che_value⟩       ::= SELF
                    | ⟨d_id⟩
                    | ⟨ch_id⟩
                    | ⟨word⟩
                    | ⟨number⟩
                    | ⟨model⟩:⟨attribute⟩
⟨ch_range⟩        ::= ⟨r_op⟩ (⟨chr_value_list⟩)
⟨r_op⟩            ::= MIN
                    | MAX
                    | AVG
⟨chr_value_list⟩  ::= ⟨chr_value⟩
                    | ⟨chr_value⟩, ⟨chr_value_list⟩
⟨chr_value⟩       ::= ⟨d_id⟩
                    | ⟨ch_id⟩
                    | ⟨model⟩:⟨attribute⟩
⟨ch_comparison⟩   ::= ⟨c_op⟩ (⟨chc_value⟩)
⟨c_op⟩            ::= = | > | <
⟨chc_value⟩       ::= ⟨ch_id⟩
                    | ⟨word⟩
                    | ⟨number⟩
                    | ⟨model⟩:⟨attribute⟩


⟨p_char⟩     ::= ⟨p_ch_id⟩, ⟨p_id⟩, ⟨p_ch_type⟩, ⟨quantity⟩,
                 ⟨conditions⟩
⟨p_ch_id⟩    ::= R_CH#⟨number⟩
⟨p_id⟩       ::= ⟨i_id⟩
                 | ⟨c_id⟩
⟨p_ch_type⟩  ::= RELEV
                 | PRIOR
                 | APPL:⟨word⟩


⟨relation⟩   ::= ⟨flat_rel⟩
                 | ⟨hier_rel⟩
                 | ⟨p_rel⟩
⟨flat_rel⟩   ::= ⟨rel_id⟩: SOURCE ⟨d_id⟩, TARGET ⟨d_id⟩,
```

```
                ⟨flat_type⟩, ⟨context⟩, ⟨conditions⟩,
                ⟨priority⟩, ⟨visibility⟩
⟨hier_rel⟩   ::= ⟨hrel_id⟩: SOURCE ⟨part⟩, TARGET ⟨part⟩,
                ⟨hier_type⟩, ⟨context⟩, ⟨conditions⟩,
                ⟨priority⟩, ⟨visibility⟩
⟨rel_id⟩     ::= R#⟨number⟩
⟨hrel_id⟩    ::= HR#⟨number⟩
⟨part⟩       ::= ⟨d_id⟩
                | ⟨hrel_id⟩
⟨flat_type⟩  ::= ELAB
                | SUMM
                | CONTEXT
                | HELP
                | SUGG
                | CHOICE
                | DEFAULT
                | APPL:⟨word ⟩
⟨hier_type⟩  ::= GROUP
                | SEQU


⟨p_rel⟩     ::= ⟨p_rel_id⟩: SOURCE ⟨r_part⟩, TARGET ⟨r_part⟩,
                ⟨hier_type⟩, ⟨conditions⟩
⟨p_rel_id⟩  ::= P_R#⟨number⟩
⟨r_part⟩    ::= ⟨p_id⟩
                | ⟨p_rel_id⟩


⟨context⟩     ::= ⟨i_id⟩
                  | ϵ
⟨priority⟩    ::= P (⟨quantity⟩, ⟨conditions⟩)
⟨visibility⟩  ::= V (⟨quantity⟩, ⟨conditions⟩)


⟨conditions⟩   ::= ⟨condition⟩
                   | NOT (⟨conditions⟩)
                   | AND (⟨conditions⟩, ⟨conditions⟩)
                   | OR (⟨conditions⟩, ⟨conditions⟩)
⟨condition⟩    ::= ⟨criterion⟩ ⟨e_op⟩ ⟨equation⟩
                   | ⟨criterion⟩ ⟨s_op⟩ (⟨enumeration⟩)
                   | ⟨criterion⟩ ⟨s_op⟩ [⟨interval⟩]
                   | ϵ
```

```
⟨criterion⟩    ::= ⟨d_id⟩
                 | ⟨model⟩:⟨attribute⟩
⟨model⟩        ::= SOCIAL_MODEL
                 | TECHNICAL_MODEL
                 | STYLISTIC_MODEL
                 | INTERACTION_MODEL
⟨attribute⟩    ::= ⟨word⟩
⟨equation⟩     ::= ⟨m_value⟩
                 | (⟨equation⟩ ⟨f_op⟩ ⟨equation⟩)
⟨enumeration⟩  ::= ⟨m_value⟩
                 | ⟨m_value⟩, ⟨enumeration⟩
⟨interval⟩     ::= ⟨value⟩, ⟨value⟩
⟨value⟩        ::= ⟨word⟩
                 | ⟨number⟩
⟨m_value⟩      ::= ⟨word⟩
                 | ⟨number⟩
                 | ⟨d_id⟩
                 | ⟨model⟩:⟨attribute⟩
⟨e_op⟩         ::= = | > | <
⟨s_op⟩         ::= IN
⟨f_op⟩         ::= + | - | * | /


⟨word⟩    ::= ⟨letter⟩
            | ⟨letter⟩⟨word⟩
⟨letter⟩  ::= a | b | c | d | e | f | g | h | i | j | k | l | m
            | n | o | p | q | r | s | t | u | v | w | x | y | z
            | A | B | C | D | E | F | G | H | I | J | K | L
            | N | O | P | Q | R | S | T | U | V | W | X | Y | Z


⟨number⟩  ::= ⟨digit⟩
            | ⟨digit⟩⟨number⟩
⟨digit⟩   ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
```

Appendix C

Syntax for Element-Based Interaction Objects

```
⟨e_object⟩ ::= E_OBJECT (
               ELEMENTS (⟨elements⟩),
               GROUPS (⟨groups⟩),
               PROPERTIES (⟨properties⟩),
               VALUES (⟨d_values⟩))


⟨elements⟩    ::= ⟨element⟩
                  | ⟨element⟩, ⟨elements⟩
⟨element⟩     ::= ⟨e_id⟩ (PROVIDE (⟨data⟩), EXTRACT (⟨data⟩),
                  SHOW (⟨data⟩))
⟨e_id⟩        ::= E#⟨number⟩
⟨data⟩        ::= ⟨d_id⟩:⟨d_type⟩
⟨d_id⟩        ::= D#⟨number⟩
⟨d_type⟩      ::= ⟨basic⟩
                  | ⟨complex⟩
⟨basic⟩       ::= STRING () | REAL () | VOID ()
⟨complex⟩     ::= SET (⟨d_type⟩)
                  | LIST (⟨d_type⟩)
                  | RECORD (⟨d_type_list⟩)
⟨d_type_list⟩ ::= ⟨d_type⟩
                  | ⟨d_type⟩, ⟨d_type_list⟩


⟨groups⟩       ::= ⟨group⟩
                   | ⟨group⟩, ⟨groups⟩
⟨group⟩        ::= ⟨g_id⟩ (⟨list⟩)
```

```
⟨g_id⟩          ::= G#⟨number⟩
⟨list⟩          ::= ⟨list_element⟩
                    | ⟨list_element⟩, ⟨list⟩
⟨list_element⟩  ::= ⟨e_id⟩
                    | ⟨g_id⟩

⟨properties⟩    ::= ⟨p_element⟩
                    | ⟨p_element⟩, ⟨properties⟩
⟨p_element⟩     ::= ⟨id⟩ (OWN (⟨p_list⟩), PUBLIC (⟨p_list⟩), INHERIT (⟨id⟩))
⟨id⟩            ::= ⟨e_id⟩
                    | ⟨g_id⟩
⟨p_list⟩        ::= ⟨p_type⟩ (VALUE (⟨value⟩), COND (⟨conditions⟩))
                    | ⟨p_type⟩ (VALUE (⟨value⟩), COND (⟨conditions⟩)), ⟨p_list⟩
⟨p_type⟩        ::= RELEV | EMPH
⟨value⟩         ::= ⟨word⟩
                    | ⟨real⟩
                    | ⟨model⟩:⟨attribute⟩
⟨conditions⟩    ::= ⟨condition⟩
                    | NOT (⟨conditions⟩)
                    | AND (⟨conditions⟩, ⟨conditions⟩)
                    | OR (⟨conditions⟩, ⟨conditions⟩)
⟨condition⟩     ::= ⟨criterion⟩ ⟨e_op⟩ ⟨equation⟩
                    | ⟨criterion⟩ ⟨s_op⟩ (⟨enumeration⟩)
                    | ⟨criterion⟩ ⟨s_op⟩ [⟨interval⟩]
                    | ϵ
⟨criterion⟩     ::= ⟨model⟩:⟨attribute⟩
⟨model⟩         ::= SOCIAL_MODEL
                    | TECHNICAL_MODEL
                    | STYLISTIC_MODEL
                    | INTERACTION_MODEL
⟨attribute⟩     ::= ⟨word⟩
⟨equation⟩      ::= ⟨value⟩
                    | (⟨equation⟩ ⟨f_op⟩ ⟨equation⟩)
⟨enumeration⟩   ::= ⟨value⟩
                ::= ⟨value⟩, ⟨enumeration⟩
⟨interval⟩      ::= ⟨value⟩, ⟨value⟩
⟨e_op⟩          ::= = | > | <
⟨s_op⟩          ::= IN
⟨f_op⟩          ::= + | - | * | /

⟨d_values⟩      ::= ⟨d_id⟩ (⟨d_value⟩)
```

```
                | ⟨d_id⟩ (⟨d_value⟩), ⟨d_values⟩
⟨d_value⟩     ::= ⟨word⟩
                | ⟨real⟩
                | SET (⟨value_list⟩)
                | LIST (⟨value_list⟩)
                | RECORD (⟨value_list⟩)
⟨value_list⟩  ::= ⟨d_value⟩
                | ⟨d_value⟩, ⟨value_list⟩


⟨number⟩  ::= ⟨digit⟩
              ⟨digit⟩⟨number⟩
⟨digit⟩   ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
⟨real⟩    ::= ⟨number⟩
              | ⟨number⟩.⟨number⟩
⟨word⟩    ::= ⟨letter⟩
              | ⟨letter⟩⟨word⟩
⟨letter⟩  ::= a | b | c | d | e | f | g | h | i | j | k | l | m
              | n | o | p | q | r | s | t | u | v | w | x | y | z
              | A | B | C | D | E | F | G | H | I | J | K | L
              | N | O | P | Q | R | S | T | U | V | W | X | Y | Z
```

Bibliography

[AA93] Paolo Atzeni and Valeria De Antonellis. *Relational Database Theory.* Benjamin/Cummings, Redwood City, CA, 1993.

[ABF+97] Jan Alexandersson, Bianka Buschbeck-Wolf, Tsutomu Fujinami, Elisabeth Maier, Norbert Reithinger, Birte Schmitz, and Melanie Siegel. Dialogue acts in VERBMOBIL-2. Verbmobil-Report 204, DFKI GmbH Saarbrücken, Universität Stuttgart, Technische Universität Berlin, Universität des Saarlandes, 1997.

[AFG+93] Elizabeth André, Wolfgang Finkler, Winfried Graf, Thomas Rist, Anne Schauder, and Wolfgang Wahlster. WIP: The automatic synthesis of multimodal presentations. In Mark T. Maybury, editor, *Intelligent Multimedia Interfaces*, pages 75–93. AAAI Press/The MIT Press, Menlo Park, CA, 1993.

[AHV95] Serge Abiteboul, Richard Hull, and Victor Vianu. *Foundations of Databases.* Addison Wesley, Reading, 1995.

[AL91] Andrea Asperti and Giuseppe Longo. *Categories, Types, and Structures: An Introduction to Category Theory for the Working Computer Scientist.* The MIT Press, Cambridge, Mass., 1991.

[Alt94] Margita Altus. Zweiter DFG-Zwischenbericht zum Projekt „Modellierung von Struktur, Semantik und Verhalten von Datenbanken: HERM+". Institut für Informatik, BTU Cottbus, 1994.

[Ano94] Anonymous. Recognizing and heeding cultural differences can be key to international business success. *Business America*, 115(10):8–11, 1994.

[Ape94] Jürgen Apel. Die Entwicklung eines Strategieberaters für den Datenbankentwurf. Master's thesis, Universität Rostock, 1994.

[Atl] Atlas Web site. URL: http://www.atlasmagazine.com/.

[Aus62] John L. Austin. *How to do Things with Words?* Oxford University Press, Oxford, 1962.

[AW74] W. Abernathy and K. Wayne. Limits of the learning curve. *Harvard Business Review*, 52(5):109–119, 1974.

[BB81] Thomas Ballmer and Waltraut Brennenstuhl. *Speech Act Classification. A Study in the Lexical Analysis of English Speech Activity Verbs.* Springer, Berlin, 1981.

[BB97] Roland Bee and Frances Bee. *Project Management. The People Challenge.* Institute of Personnel and Development, London, 1997.

[BBD+93] M. Bauer, S. Biundo, D. Dengler, J. Koehler, and G. Merziger. PHI — a logic-based tool for intelligent help systems. In *Proc. of IJCAI-93*, Chambéry, France, pages 460–466. 1993.

[BCN92] Carlo Batini, Stefano Ceri, and Shamkant B. Navathe. *Conceptual Database Design.* Benjamin/Cummings, Redwood City, CA, 1992.

[BFJ96] Hans-Jörg Bullinger, Klaus-Peter Fähnrich, and Christian Janssen. Ein Beschreibungskonzept für Dialogabläufe bei graphischen Benutzungsschnittstellen. *Informatik Forschung und Entwicklung*, 11(2):84–93, 1996.

[BHLV95] François Bodart, Anne-Marie Hennebert, Jean-Marie Leheureux, and Jean Vanderdonckt. A model-based approach to presentation: A continuum from task analysis to prototype. In Fabio Paternò, editor, *Proc. of the Eurographics Workshop on Design, Specification, and Verification of Interactive Systems*, Bocca di Magra, pages 77–94. Springer, Berlin, 1995.

[Boe88] Barry W. Boehm. A spiral model of software development and enhancement. *IEEE Computer*, 21(6):61–72, 1988.

[BR95] Hans Burg and Reind P. van de Riet. The impact of linguistics on conceptual models: Consistency and understandability. In Mokrane Bouzeghoub and Elisabeth Métais, editors, *Proc. of the First International Workshop on Applications of Natural Language to Databases, NLDB'95*, Versailles, France. AFCET, 1995.

[Bra92] Marshall Brain. *Motif Programming: The Essentials—and More.* Digital Press, 1992.

[Bru95] Peter Brusilovsky. Integrating hypermedia and intelligent tutoring technologies: From systems to authoring tools. In Nigel Major, Tom Murray, and Charles Bloom, editors, *Proc. of the AI-ED-95 Workshop on Authoring Shells for Intelligent Tutoring Systems*, Washington, DC. Online proceedings, URL: http://www.pitt.edu/~al/aied/workshop.html, 1995.

[Bru96] Peter Brusilovsky. Methods and techniques of adaptive hypermedia. *User Modeling and User Adapted Interaction, Special Issue on Adaptive Hypertext and Hypermedia*, 6(2-3):87–129, 1996.

[BS97] Peter Brusilovsky and Elmar Schwarz. User as student: Towards an adaptive interface for advanced web-based applications. In Anthony Jameson, Cécile Paris, and Carlo Tasso, editors, *User Modeling: Proc. of the Sixth International Conference, UM97*, Chia Laguna, Italy, pages 177–188. Springer, New York, 1997.

[Büh82] Karl Bühler. *Sprachtheorie*. Gustav Fischer Verlag, Stuttgart, 1982.

[Cam99] Peter J. Cameron. *Sets, Logic and Categories*. Springer, London, 1999.

[Car94] Luiza M. F. Carneiro-Coffin. *JASMINUM: Joining ADVs and state machines in a notation for user-interface modelling*. PhD thesis, University of Waterloo, Waterloo, Ontario, Canada, 1994.

[CCC+96] Tiziana Catarci, Shi-Kuo Chang, Maria F. Costabile, Stefano Levialdi, and Giuseppe Santucci. A graph-based framework for multiparadigmatic visual access to databases. *IEEE Transactions on Knowledge and Data Engineering*, 8(3):455–475, 1996.

[CCK+94] Richard C. H. Connor, Quintin I. Cutts, Graham N. C. Kirby, V. S. Moore, and Ronald Morrison. Unifying interaction with persistent data and program. In Peter Sawyer, editor, *Interfaces to Database Systems, Proc. of the 2nd International Workshop on User Interfaces to Databases, IDS94*, Lancaster, pages 197–212. Springer, Berlin, 1994.

[CCL94] Luiza M. F. Carneiro-Coffin, Donald D. Cowan, and Carlos J. P. Lucena. ADVcharts: a visual formalism for interactive systems. *SIGCHI Bulletin*, 26(2):74–77, 1994.

[CCM+96] Tiziana Catarci, Maria F. Costabile, Antonio Massari, Lorenzo Saladini, and Giuseppe Santucci. A multiparadigmatic environment for

interacting with databases. *ACM SIGCHI Bulletin, Special Issue on the Role of HCI in Italy*, 28(3):89–96, July 1996.

[CL99] Wolfram Clauß and Jana Lewerenz. Abstract interaction specification for information services. In Mehdi Khosrowpour, editor, *Managing Information Technology Resources in Organizations in the Next Millenium. Proc. of the 1999 IRMA International Conference*, Hershey, PA, pages 89–96. Idea Group Publishing, Hershey, PA, 1999.

[Cla00] Wolfram Clauß. *Entwicklung interaktiver Transaktionssysteme: Herstellung, Wartung und Wiederverwendung*. PhD thesis, BTU Cottbus, 2000. In preparation.

[CLS99a] Wolfram Clauß, Jana Lewerenz, and Kati Seelig. Paradigmenunabhängige Konzepte für die Dialogverwaltung in Informationssystemen. In Jörg Desel, Klaus Pohl, and Andy Schürr, editors, *Modellierung'99, Workshop der Gesellschaft für Informatik*, Karlsruhe, pages 123–138. Teubner, Stuttgart, 1999.

[CLS99b] Wolfram Clauß, Jana Lewerenz, and Srinath Srinivasa. Paradigm-independent modeling concepts and translations for interactive information systems. Informatik-Bericht I-04/1999, BTU Cottbus, 1999.

[CLT97] Wolfram Clauß, Jana Lewerenz, and Bernhard Thalheim. Dynamic dialog management. In Stephen W. Liddle, editor, *Proc. of the ER'97 Workshop on Behavioral Models and Design Transformations*, Los Angeles. Online proceedings, URL: http://osm7.cs.byu.edu/ER97/workshop4/, 1997.

[Col91] *Collins English dictionary*. HarperCollins Publishers, Glasgow, 3rd edition, 1991.

[Cou87] Joëlle Coutaz. PAC, an object oriented model for dialog design. In Hans-Jörg Bullinger and Brian Shackel, editors, *Human-Computer Interaction – INTERACT'87. Proc. of the Second IFIP Conference on Human-Computer Interaction*, Stuttgart, Germany, pages 431–436. North-Holland, Amsterdam, 1987.

[Cou95] Joëlle Coutaz. Evaluation techniques: Exploring the intersection of HCI and software engineering. In Richard N. Taylor and Joëlle Coutaz, editors, *Proc. of the ICSE'94 Workshop on Software Engineering and Human-Computer Interaction*, Sorrento, Italy, pages 35–48. Springer, Berlin, 1995.

[CSS] Cascading StyleSheets Web site. URL: http://www.w3.org/Style/CSS/.

[CT95] Tiziana Catarci and Laura Tarantino. A hypergraph-based framework for visual interaction with databases. *Journal of Visual Languages and Computing*, 6(2):135–166, 1995.

[CT97] Wolfram Clauß and Bernhard Thalheim. Abstraction layered structure-process codesign. In D. Janaki Ram, editor, *Proc. of the 8th International Conference on Management of Data*, Madras, India, pages 223–237. Narosa Publishing House, New Delhi, 1997.

[CW93] Kevin Cox and David Walker. *User-Interface Design.* Prentice Hall, New York, 2nd edition, 1993.

[DFAB93] Alan Dix, Janet Finlay, Gregory Abowd, and Russell Beale. *Human-Computer Interaction.* Prentice Hall, New York, 1993.

[DJPQ94] Oscar Díaz, Arturo Jaime, Norman W. Paton, and Ghassan al-Qaimari. Supporting dynamic displays using active rules. *SIGMOD Record*, 23(1):21–26, 1994.

[DMKS93] Hartmut Dieterich, Uwe Malinowski, Thomas Kühme, and Matthias Schneider-Hufschmidt. State of the art in adaptive user interfaces. In Matthias Schneider-Hufschmidt, Thomas Kühme, and Uwe Malinowski, editors, *Adaptive User Interfaces: Principles and Practise*, pages 13–48. Elsevier Science, Amsterdam, 1993.

[Dow91] Andy Downton, editor. *Engineering the Human-Computer Interface.* McGraw-Hill, London, 1991.

[Düs97] Antje Düsterhöft. *Zur natürlichsprachlichen interaktiven Unterstützung im Datenbankentwurf.* Infix, Sankt Augustin, 1997.

[DW91] Stephen W. Draper and Kevin W. Waite. Iconographer as a visual programming system. In Dan Diaper and Nick Hammond, editors, *People and Computers VI: Proc. of HCI'91*, Cambridge, pages 171–185. Cambridge University Press, Cambridge, UK, 1991.

[Ebe94] Ray E. Eberts. *User interface design.* Prentice Hall, Englewood Cliffs, 1994.

[Eco85] Umberto Eco. *Semiotics and the philosophy of language.* Indiana University Press, Bloomington, 1985.

[Eco97] Umberto Eco. *A theory of semiotics.* Indiana University Press, Bloomington, 1997.

[EZ96] John Eklund and Romain Zeilinger. Navigating the web: Possibilities and practicalities for adaptive navigational support. In *Proc. of the Second Australian World Wide Web Conference*, Gold Coast, Australia. 1996.

[Fac92] Hubert G. Fackeldey. *Norm und Begründung: zur Logik normativen Argumentierens.* Peter Lang, Bern, 1992.

[FKN97a] Josef Fink, Alfred Kobsa, and Andreas Nill. Adaptable and adaptive information access for all users including the disabled and the elderly. In Anthony Jameson, Cécile Paris, and Carlo Tasso, editors, *User Modeling: Proc. of the Sixth International Conference, UM97*, pages 171–173. Springer, New York, 1997.

[FKN97b] Josef Fink, Alfred Kobsa, and Andreas Nill. Benutzerorientierte Adaptivität und Adaptierbarkeit im Projekt AVANTI. In R. Liskowski, B. M. Velichkovsky, and W. Wünschmann, editors, *Proc. of Software-Ergonomie 97. Usability Engineering: Integration von Mensch-Computer-Interaktion und Software-Entwicklung*, Dresden, pages 135–143. Teubner, Stuttgart, 1997.

[FKS95] Babak Amin Farshchian, John Krogstie, and Arne Sølvberg. Integration of user interface and conceptual modeling. In Constantine Stephanidis, editor, *Proc. of the 1st ERCIM Workshop 'Towards User Interfaces for All'*, Heraklion, Greece. 1995.

[FL00] Paul Foley and Jana Lewerenz. From black model T Ford to chameleon. A model of interactive communication to facilitate relationship marketing on the Internet. In *Proc. of the First Annual Global Information Technology Management World Conference, GITM.* Memphis, 2000. To appear.

[FMS94] Markus Fischer, Elisabeth Maier, and Adelheit Stein. Generating cooperative system responses in information retrieval dialogues. In Robin Fawcett, Eduard Hovy, David McDonald, Marie Meteer, Donia Scott, and Koenraad deSmedt, editors, *Proc. of the 7th International Workshop on Natural Language Generation*, Kennebunkport, Maine, pages 207–216. 1994.

[Fra97] Norman Fraser. Assessment of interactive systems. In Dafydd Gibbon, Roger Moore, and Richard Winski, editors, *Handbook of Standards*

and Resources for Spoken Language Systems. Mouton de Gruyter, New York, 1997.

[FST98] Thomas Feyer, Klaus-Dieter Schewe, and Bernhard Thalheim. Conceptual design and development of information services. In Tok Wang Ling, Sudha Ram, and Mong-Li Lee, editors, *Proc. of the 17th International Conference on Conceptual Modeling, ER'98*, Singapore, pages 7–20. Springer, 1998.

[Fur86] George W. Furnas. Generalized fisheye views. In *Proc. of the 1986 Conference on Human Factors in Computing Systems, CHI'86*, Boston, pages 16–23. ACM Press, New York, 1986.

[GMP+98] Tony Griffiths, Jo McKirdy, Norman Paton, Jessie Kennedy, Richard Cooper, Peter Barclay, Carole Goble, Philip Gray, Michael Smyth, Adrian West, and Andrew Dinn. An open model-based interface development system: The Teallach approach. In Panos Markopoulos and Peter Johnson, editors, *Proc. of the 5th Eurographics Workshop on Design, Specification and Verification of Interactive Systems, DSV-IS*, Abingdon, UK, pages 32–49. Springer, Vienna, 1998.

[Gra97] Philip D. Gray. The Representer system. Online manuscript, URL: http://www.dcs.gla.ac.uk/~pdg/research/representer/representer.html, March 1997.

[Gre86] Mark Green. A survey of three dialogue models. *ACM Transactions on Graphics*, 5(3):244–275, 1986.

[Gri75] Herbert P. Grice. Logic and conversation. In Peter Cole and John L. Morgan, editors, *Syntax and Semantics*, Volume 3, pages 41–58. Academic Press, New York, 1975.

[GWD90] Philip D. Gray, Kevin W. Waite, and Stephen W. Draper. Do-it-yourself iconic displays: Reconfigurable iconic representations of application objects. In Dan Diaper et al., editors, *Human-Computer Interaction – INTERACT'90. Proc. of the Third IFIP Conference on Human-Computer Interaction*, Cambridge, UK, pages 639–644. North-Holland, Amsterdam, 1990.

[HCM+83] Wolfgang Hoeppner, Thomas Christaller, Heinz Marburger, Katharina Morik, Bernhard Nebel, Mike O'Leary, and Wolfgang Wahlster. Beyond domain-independence: Experience with the development of a German natural language access system to highly diverse background systems. In Alan Bundy, editor, *Proc. of the Eigth International Joint*

Conference on Artificial Intelligence, IJCAI'83, Karlsruhe, Germany, pages 588–594. William Kaufmann, 1983.

[Hel91] Dan Heller. *XView Programming Manual.* O'Reilly, Sebastopol, CA, 3rd edition, 1991.

[Hop93] H. Ulrich Hoppe. Intelligent user support based on task models. In Matthias Schneider-Hufschmidt, Thomas Kühme, and Uwe Malinowski, editors, *Adaptive User Interfaces: Principles and Practise*, pages 167–181. Elsevier Science, Amsterdam, 1993.

[HS94] Elizabeth A. Hinkelman and Stephen P. Spackman. Communicating with multiple agents. In *Proc. of the 15th International Conference on Computational Linguistics, COLING-94*, Kyoto, Japan. 1994.

[Hud87] Scott Hudson. UIMS support for direct manipulation interfaces. *ACM Computer Graphics*, 21(2):120–124, 1987.

[JWZ93] Christian Janssen, Anette Weisbecker, and Jürgen Ziegler. Generating user interfaces from data models and dialog net specifications. In Stacey Ashlund, Kevin Mullet, Austin Henderson, Erik Hollnagel, and Ted White, editors, *Proc. of the 1993 Conference on Human Factors in Computing Systems, INTERCHI'93*, Amsterdam, pages 418–423. ACM Press, New York, 1993.

[Kay93] Judy Kay. Pragmatic user modeling for adaptive interfaces. In Matthias Schneider-Hufschmidt, Thomas Kühme, and Uwe Malinowski, editors, *Adaptive User Interfaces: Principles and Practise*, pages 129–148. Elsevier Science, Amsterdam, 1993.

[KL96] Gunther Kress and Theo van Leeuwen. *Reading Images: The Grammar of Visual Design.* Routledge, London, 1996.

[Kob93] Alfred Kobsa. User modeling: Recent work, prospects and hazards. In Matthias Schneider-Hufschmidt, Thomas Kühme, and Uwe Malinowski, editors, *Adaptive User Interfaces: Principles and Practise.* Elsevier Science, Amsterdam, 1993.

[KP88] Glenn Krasner and Stephen Pope. A cookbook for using the Model View Controller user interface paradigm in Smalltalk-80. *Journal of Object-Oriented Programming*, 1(3):26–49, 1988.

[KPL95] Paul Kahn, Ronnie Peters, and George P. Landow. Three fundamental elements of visual rhetoric in hypertext. In Wolfgang Schuler, Jörg Hannemann, and Norbert Streitz, editors, *Designing User Interfaces for Hypermedia.* Springer, Berlin, 1995.

[Las48] Harold Lasswell. The structure and function of communication in society. In L. Bryson, editor, *The Communication of Ideas*, pages 37–51. Institute for Religious and Social Studies, New York, 1948.

[Lev83] Stephen C. Levinson. *Pragmatics*. Cambridge University Press, Cambridge, UK, 1983.

[Lew96] Jana Lewerenz. Entwicklung einer Benutzungsmodellierungskomponente zur Gestaltung von natürlichsprachlichen Dialogen für RADD. Master's thesis, Universität Rostock, 1996.

[Lew99] Jana Lewerenz. On the use of natural language concepts for the conceptual modeling of interaction in information systems. In Günther Fliedl and Heinrich C. Mayr, editors, *Proc. of the 4th International Conference on Applications of Natural Language to Databases, NLDB'99*, Klagenfurt, Austria, pages 61–75. Österreichische Computer Gesellschaft, 1999.

[LN99] Jonathan K. Lazar and Anthony F. Norcio. To err or not to err, that is the question: Novice user perception of errors while surfing the web. In Mehdi Khosrowpour, editor, *Managing Information Technology Resources in Organizations in the Next Millenium. Proc. of the 1999 IRMA International Conference*, Hershey, PA, pages 321–325. Idea Group Publishing, Hershey, PA, 1999.

[LO94] Mark W. Lansdale and Thomas C. Ormerod. *Understanding Interfaces: A Handbook of Human-Computer Dialogue*. Academic Press, London, 1994.

[LS00] Jana Lewerenz and Srinath Srinivasa. Abstraction of the interaction properties of an information system for achieving target platform independence. In Ulrich Frank and Jürgen Ebert, editors, *Modelle und Modellierungssprachen in Informatik und Wirtschaftsinformatik: Beiträge des Workshops Modellierung 2000 – Workshop der Gesellschaft für Informatik*, St. Goar, pages 5–17. Fölbach-Verlag, Koblenz, 2000.

[Łuk90] Witold Łukaszewicz. *Non-monotonic reasoning: formalization of commonsense reasoning*. Ellis Norwood, New York, 1990.

[McT98] Michael McTear. Spoken dialogue technology: Enabling the conversational user interface. Online manuscript, URL: http://www.infj.ulst.ac.uk/~cbdg23/survey/spoken_dialogue_technology.html, 1998.

[MHC+96] Brad Myers, Jim Hollan, Isabel Cruz, Steve Bryson, Dick Bulterman, Tiziana Catarci, Wayne Citrin, Ephraim Glinert, Jonathan Grudin, and Yannis Ioannidis. Strategic directions in human-computer interaction. *ACM Computing Surveys*, 28(4):794–809, 1996.

[Mit90] Deborah A. Mitta. A fisheye presentation strategy: Aircraft maintenance data. In Dan Diaper et al., editors, *Human-Computer Interaction – INTERACT'90. Proc. of the Third IFIP Conference on Human-Computer Interaction*, Cambridge, UK. North-Holland, Amsterdam, 1990.

[MKB95] Kenneth J. Mitchell, Jessie B. Kennedy, and Peter J. Barclay. Using a conceptual language to describe a database and its interface. In John A. Keane and Carole A. Goble, editors, *Advances in Databases, Proc. of the 13th British National Conference on Databases*, Manchester, pages 101–119. Springer, Berlin, 1995.

[MKB96] Kenneth J. Mitchell, Jessie B. Kennedy, and Peter J. Barclay. A framework for user-interfaces to databases. In Tiziana Catarci, Maria F. Costabile, Stefano Levialdi, and Giuseppe Santucci, editors, *Proc. of the International Workshop on Advanced Visual Interfaces, AVI'96*, Gubbio, Italy, pages 81–90. ACM Press, New York, 1996.

[MR92] Brad A. Myers and Mary Beth Rosson. Survey on user interface programming. In *Proc. of the 1992 Conference on Human Factors in Computing Systems, CHI'92*, Monterey, CA, pages 195–202. ACM Press, New York, 1992.

[MS95] Kevin Mullet and Darrell Sano. *Designing Visual Interfaces.* SunSoft Press, Mountain View, CA, 1995.

[MT87] William C. Mann and Sandra A. Thompson. Rhetorical structure theory: A theory of text organization. In Livia Polanyi, editor, *The Structure of Discourse.* Ablex, Norwood, N. J., 1987.

[Mye93] Brad A. Myers. Why are human-computer interfaces difficult to design and implement? Technical Report CMU-CS-93-183, Carnegie Mellon University, 1993.

[Nad88] Mihai Nadin. Interface design: A semiotic approach. *Semiotica*, 69(3/4):269–302, 1988.

[Nie93] Jakob Nielsen. *Usability Engineering.* AP Professional, Boston, 1993.

[Nie98] Jakob Nielsen. The increasing conservatism of web users. Jakob Nielsen's Alertbox from March 22, 1998. URL: http://www.useit.com/alertbox/980322.html, 1998.

[NST94] Clifford Nass, Jonathan Steuer, and Ellen R. Tauber. Computers are social actors. In *Celebrating Interdependence: Proc. of the 1994 Conference on Human Factors in Computing Systems, CHI'94*, Boston, pages 72–78. ACM Press, New York, 1994.

[PB93] Gustav Pomberger and Günther Blaschek. *Grundlagen des Software Engineering: Prototyping und objektorientierte Software-Entwicklung.* Hanser, Munich, 1993.

[PCOM99] Angel R. Puerta, Eric Cheng, Tunhow Ou, and Justin Min. MOBILE: User-centered interface building. In *Proc. of the 1999 Conference on Human Factors in Computing Systems, CHI'99*, Pittsburgh, pages 426–433. ACM Press, New York, 1999.

[PDDJ96] Norman W. Paton, Khoa Doan, Oscar Díaz, and Arturo Jaime. Exploitation of object-oriented and active constructs in database interface development. In Jessie Kennedy and Peter J. Barclay, editors, *Proc. of the 3rd International Workshop on Interfaces to Database Systems*, Electronic Workshops in Computing, Edinburgh, UK. Springer, 1996.

[PH85] Günther P. Pfaff and Paul J. W. ten Hagen, editors. *User Interface Management Systems.* Springer, Berlin, 1985.

[Poi92] Axel Poigné. Basic category theory. In Samson Abramsky, Dov M. Gabbay, and Thomas S. E. Maibaum, editors, *Handbook of Logic in Computer Science*, Volume 1 Background: Mathematical Structures, pages 413–640. Clarendon Press, Oxford, 1992.

[Pol92] Martha E. Pollack. The uses of plans. *Artificial Intelligence*, 57(1):43–68, 1992.

[Pol96] Martha E. Pollack. Planning in dynamic environments: The DIPART system. In A. Tate, editor, *Advanced Planning Technology.* AAAI Press, 1996.

[PTHM90] Franz Penz, Manfred Tscheligi, Günter Haring, and Martina Manhartsberger. The power of parameterisable objects in modern user interfaces. In Dan Diaper et al., editors, *Human-Computer Interaction – INTERACT'90. Proc. of the Third IFIP Conference*

on Human-Computer Interaction, Cambridge, UK, pages 469–472. North-Holland, Amsterdam, 1990.

[Rad99] Michael Radigk. Strukturorientierte Generierung von graphischen Benutzerschnittstellen für Datenbankanwendungen, 1999. Semester Thesis, BTU Cottbus.

[Rad00] Michael Radigk. Darstellung und Anpassung von Interaktionsobjekten in der Co-Design-Umgebung. Master's thesis, BTU Cottbus, 2000. In preparation.

[RB99] Andreas Rauber and Harald Bina. A metaphor graphics based representation of digital libraries on the World Wide Web: Using the libViewer to make metadata visible. In *Proc. of the DEXA'99 Workshop on Web-based Information Visualization, WebVis'99*, Florence, Italy. IEEE Computer Society Press, New York, 1999.

[Ric79] Elaine Rich. User modeling via stereotypes. *Cognitive Science*, 3:329–354, 1979.

[RM99] Andreas Rauber and Dieter Merkl. The SOMLib digital library system. In Serge Abiteboul and Anne-Marie Vercoustre, editors, *Proc. of the Third European Conference on Digital Libraries, ECDL'99*, Paris, pages 323–342. Springer, 1999.

[Roy70] Winston W. Royce. Managing the development of large software systems: Concepts and techniques. In *Proc. WESCON Western Electronic Show and Convention*, Los Angeles, pages 1–9. 1970.

[RR98] Simon R. Rollinson and Stuart A. Roberts. Formalizing the informational content of database user interfaces. In Tok Wang Ling, Sudha Ram, and Mong-Li Lee, editors, *Proc. of the 17th International Conference on Conceptual Modeling, ER'98*, Singapore, pages 65–77. Springer, 1998.

[SA91] Volker Sperschneider and Griogorios Antoniou. *Logic: A foundation for Computer Science*. Addison-Wesley, Wokingham, 1991.

[Sau16] Ferdinand de Saussure. *Course de linguistique général*. Payot, Paris, 1916.

[Sch96] Bettina Schewe. *Kooperative Softwareentwicklung: ein objektorientierter Ansatz*. Deutscher Universitäts-Verlag, Wiesbaden, 1996.

[SE95] Egbert Schlungbaum and Thomas Elwert. TADEUS — a model-based approach to the development of interactive software systems. Rostocker Informatik–Berichte, Universität Rostock, 1995.

[SE96] Egbert Schlungbaum and Thomas Elwert. Automatic user interface generation from declarative models. In J. Vanderdonckt, editor, *Proc. of the 2nd International Workshop on Computer-Aided Design of User Interfaces, CADUI'96*, Namur, pages 3–18. Presses Universitaires de Namur, 1996.

[Sea69] John A. Searle. *Speech Acts: An Essay in the Philosophy of Language.* Cambridge University Press, Cambridge, UK, 1969.

[See98] Kati Seelig. Zielplattformunabhängige Konzepte für die Dialogverwaltung in einem integrativen Entwurfsmodell, 1998. Semester Thesis, BTU Cottbus.

[See99] Kati Seelig. Prozeßspezifikationen für interaktive Datenbankanwendungen. Master's thesis, BTU Cottbus, 1999.

[SFW99] Myra Spiliopoulou, Lukas Faulstich, and Karsten Winkler. A data miner analyzing the navigational behaviour of Web users. In *Proc. of the Workshop on Machine Learning in User Modelling of the ACAI'99 International Conference*, Chania, Greece. 1999.

[Sha48] Claude Elwood Shannon. A mathematical theory of communication. *Bell Systems Technical Journal*, 27:379–423 and 623–656, July and October 1948.

[SKM93] Matthias Schneider-Hufschmidt, Thomas Kühme, and Uwe Malinowski, editors. *Adaptive User Interfaces: Principles and Practise.* Elsevier Science, Amsterdam, 1993.

[SLN92] Pedro Szekely, Ping Luo, and Robert Neches. Facilitating the exploration of interface design alternatives: The HUMANOID model of interface design. In *Proc. of the 1992 Conference on Human Factors in Computing Systems, CHI'92*, Monterey, CA, pages 507–515. ACM Press, New York, 1992.

[Smi99] Richard Smith. The RealJukeBox monitoring system. Online manuscript, URL: http://www.tiac.net/users/smiths/privacy/realjb.htm, October 1999.

[Som95] Ian Sommerville. *Software Engineering.* Addison-Wesley, Harlow, 5th edition, 1995.

[SS95] Anthony Savidis and Constantine Stephanidis. Developing dual user interfaces for integrating blind and sighted users: the HOMER UIMS. In *Proc. of the 1995 Conference on Human Factors in Computing Systems, CHI'95*, Denver, CO, pages 106–113. ACM Press, New York, 1995.

[SS00] Klaus-Dieter Schewe and Bettina Schewe. Integrating database and dialogue design. *Knowledge and Information Systems*, 2(1), 2000.

[SSR93] Graeme Shanks, Graeme Simsion, and M. Rembach. The role of experience in conceptual data modelling. In *Proc. of the 4th Australian Conference on Information Systems*, Brisbane, Australia, pages 365–378. 1993.

[Ste99] Roger Stewart, editor. *Gower Handbook of Teamworking.* Gower, Brookfield, 1999.

[Tha97] Bernhard Thalheim. Codesign von Struktur, Funktionen und Oberflächen von Datenbanken. Informatik-Bericht I-05/1997, BTU Cottbus, 1997.

[Tha00] Bernhard Thalheim. *Fundamentals of entity-relationship modeling.* Springer, 2000. To appear.

[UIM92] The UIMS Tool Developers Workshop. A metamodel for the run-time architecture of interactive systems. *SIGCHI Bulletin*, 24(1):32–37, 1992.

[Urb91] Waldemar Urbanek. *Software-Ergonomie und benutzerangemessene Auswahl von Werkzeugen bei der Dialoggestaltung.* de Gruyter, Berlin, 1991.

[Val82] Eugene Vale. *The Technique of Screen & Television Writing.* Simon & Schuster, New York, 1982.

[Wah93] Wolfgang Wahlster. Verbmobil - translation of face-to-face dialogs. In *Proc. of the 4th Machine Translation Summit*, Kobe, Japan, pages 127–135. 1993.

[Weg97] Peter Wegner. Why interaction is more powerful than algorithms. *Communications of the ACM*, 40(5):81–91, 1997.

[Wei95] Anette Weisbecker. *Ein Verfahren zur automatischen Generierung von software-ergonomisch gestalteten Benutzungsoberflächen.* Springer, Berlin, 1995.

[WG98] Peter Wegner and Dina Goldin. Interaction as a framework for modeling. In P. P. Chen, Jacky Akoka, Hannu Kangassalo, and Bernhard Thalheim, editors, *Conceptual Modeling: Current Issues and Future Directions*. Springer, Berlin, 1998.

[WG99] Peter Wegner and Dina Goldin. Mathematical models of interactive computing. Brown Technical Report CS 99-13, Brown University, 1999.

[Who56] Benjamin Lee Whorf. Science and linguistics. In John B. Carroll, editor, *Language, Thought, and Reality: Selected Writings of Benjamin Lee Whorf*, pages 207–219. MIT Press, Cambridge, Mass., 1956.

[WL98] Lynda Weinman and Jon Lentz. *Deconstructing Web Graphics 2*. New Riders Publishing, Macmillan, 1998.

[Wri77] Georg Henrik von Wright. *Handlung, Norm und Intention. Untersuchungen zur deontischen Logik*. de Gruyter, Berlin, 1977.

[WSA+96] Peter J. Wyard, Alison D. Simons, Steve Appelby, Edward Kaneen, Sandra H. Williams, and Keith R. Preston. Spoken language systems - beyond prompt and response. *BT Technology Journal*, 14(1):187–205, 1996.

[WWW] WWWebster Dictionary Pronunciation Guide. URL: http://www.m-w.com/pronguid.htm.

[YM94] R. Michael Young and Johanna D. Moore. DPOCL: A principled approach to discourse planning. In Robin Fawcett, Eduard Hovy, David McDonald, Marie Meteer, Donia Scott, and Koenraad deSmedt, editors, *Proc. of the 7th International Workshop on Natural Language Generation*, Kennebunkport, Maine, pages 13–20. 1994.

[ZXH98] Osmar Zaïane, Man Xin, and Jiawei Han. Discovering Web access patterns and trends by applying OLAP and data mining technology on Web logs. In *Proc. of the Advances in Digital Libraries Conference, ADL'98*, Santa Barbara, CA, pages 19–29. IEEE Computer Society Press, New York, 1998.